REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
11-Jan-2000	Annual Report	14-11-98 to 13-11-99

4. TITLE AND SUBTITLE

Annual Progress Report No. 10

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N00014-99-1-0158

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Reifsnider, Kenneth,
Virginia Polytechnic Institute and State University

5d. PROJECT NUMBER

5e. TASK NUMBER
Data item #0034

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Virginia Polytechnic Institue and State University,
Blacksburg, VA 24061

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR, 800 North Quincy Street, Arlington, VA 22217-5660

10. SPONSOR/MONITOR'S ACRONYM(S)
ONR 311

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Annual Report No. 10 for the NAVCIITI Program.

20000114 035

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE		43	Dr. Kenneth L. Reifsnider
					19b. TELEPONE NUMBER *(Include area code)* (540) 231-9359

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI-Std Z39-18

Navy Collaborative Integrated Information Technology Initiative

(NAVCIITI)

ONR Grant N00014-99-1-0158

Report No. 10

360 DAC

Submitted to:

Mr. Paul Quinn
ONR 311
Office of Naval Research
Ballston Centre Tower One
800 North Quincy Street
Arlington VA 22217-5660

Submitted by:

Virginia Tech
Blacksburg VA 24061

14 November 1998 – 13 November 1999

Navy Collaborative Integrated Information Technology Initiative (NAVCIITI)

This is the tenth report of the NAVCIITI program, that covers the 360 Days After Contract (DAC) period. This report is submitted as the annual report.

1. Introduction

The long-term objective of the NAVCIITI program is to provide strong, integrated research capabilities in broad user communications testbeds, systems engineering, fiber optic sensors and transmission devices, secure and reliable wireless communications, effective user-friendly human computer interfaces, and scientific visualization to the Navy community. This initiative will improve the Navy's capabilities to support distributed computing, integrated services training, education, information dissemination, and simulation.

The multiyear program will establish a Navy Collaborative Integrated Information Technology Initiative (NAVCIITI, pronounced "NAV city") by creating an Advanced Communication and Information Technology Center (ACITC), on campus at Virginia Tech, integrating and leveraging the efforts of more than 60 investigators currently under contract to the Navy by providing equipment and facilities for their effort, and using the collective capabilities of NAVCIITI to support Navy initiatives in distributed computing, integrated services training, education, information dissemination, and simulation, especially for purposes of network-centric battle management, managing and maintaining C4ISR attributes, and enhancement of the Naval intranet. The scope of the proposed program was developed as a result of discussions, and briefings with a group of Navy unit leaders.

2.0 Technical Summary

The program has now completed eleven months. Major enabling equipment purchases have been made, as outlined briefly in our fourth monthly report, and as updated in the sixth monthly report. We have now made significant research progress using this new equipment and related facilities on campus.

The vision of the program is to provide the precursor of development, demonstration, and configuration of high payoff technologies in support of the Navy's Command & Control vision. The NAVCIITI framework is based on leveraging and capitalizing university's expertise and ongoing research in the thrust technologies and engineering in general to establish the architecture and the interoperability of the Navy's Intranet for battle-space management. The focus of the program is to build a prototype test bed for a Network Centric Environment (NCE) in support of C4ISRT (Command, Control, Communication, Computing, Intelligence, Surveillance, Reconnaissance, and Targeting). The NAVCIITI program consists of several projects covering several thrust areas of research. Wireless secure communication will be used in NAVCIITI to implement wireless secure LAN, wearable computers, PDAs (personal digital assistant) interoperability with VON (Virtual Operations Network), and interaction with other platforms. Wideband smart antennas form the front end of the network; the benefits of these devices derive from their ability to conserve spectrum, improve the network connectivity and reduce weight and size of the hardware. Modeling and simulation will be performed to determine the metrics of Command & Control. The contributions of C4ISRT functionalities in the NCE will be evaluated through mathematical modeling, visualization, communication, and collaboration tools. Platforms (ships, aircraft, submarines are critical nodes in the NCE. NAVCIITI will construct a test bed for the design and implementation of a Ship Information Management System (SIMS).

Progress in specific areas is discussed in the following sections.

2.1 Implementation Plan for Reconfigurable Software Radio (J. Reed, P. Athanas, R. Boyle, S.Srikanteswara –Task 1.2.1) CDRL Data Item 0006

Implementation Architecture for Soft Radio Encoding

Srikathyayani Srikanteswara, Dr. Jeffrey H. Reed, Dr. Peter Athanas, Dr. Robert Boyle

November 19, 1999

Abstract

While many soft/software radio architectures have been suggested and implemented, there remains a lack of a formal design methodology that can be used to design and implement these radios. This paper presents a unified architecture for the design of soft radios on a reconfigurable platform called the *Layered Radio Architecture*. The layered architecture makes it possible to incorporate all of the features of a software radio while minimizing complexity issues. The layered architecture also defines the methodology for incorporating changes and updates into the system. An example implementation of the layered architecture on actual hardware is presented in the paper.

1 Introduction to Soft Radios

Software radios are evolving as flexible all-purpose radios that can implement new and different standards or protocols through reprogramming [9, 3, 8]. Software radios give rise to the possibility of having multi-mode terminals without the "Velcro approach" of including separate silicon for each possible standard. Software radios can reduce the cost of manufacturing and testing, while providing a quick way to upgrade the product to take advantage of newer signal processing techniques and new wireless phone applications. The ideal software radio has a set of features that are not yet realizable in commercial systems due to limitations in current technology and cost considerations. Specifically, DSP microprocessors are not fast enough to implement all radio functions and thus some dedicated Application Specific Integrated Circuits (ASIC) are required. This is a consequence of processing speed being traded for flexibility and efficiency.

In this paper the term *soft radio* denotes a completely configurable radio that can be programmed in software to reconfigure physical hardware. In other words, the same piece of hardware can be modified to perform different functions at different times. This allows the hardware to be specifically tailored to the application at hand, resulting in greatly increased speed and silicon efficiency, while maintaining a high degree of flexibility.

Although most researchers agree on the advantages of the soft radio, there is a lack of a general design methodology for soft radio architectures. Most practical implementations are based on ad-hoc approaches that can only be used for few specific systems. In this paper a formal architecture for soft radios called the *layered radio architecture* is developed for reconfigurable platforms.

2 Design Issues in a Soft Radio

A number of papers describe various software radio architectures and design considerations [1, 9, 3, 8]. The key design issues include design of fast and efficient A/D converters, flexibility at the RF front end, effective data management procedures, resource allocation and smooth reconfigurability of the hardware. The A/D and the flexible RF front end are extremely challenging to design, particularly for a handset, and set the limitations of dynamic range and bandwidth. The design issues involving the A/D converter and the RF design are beyond the scope of this paper but are covered in [11, 5].

A soft radio is a complex entity that needs to handle very high data rates efficiently. To process data efficiently entails good computing resource allocation. There are very important differences in the resource allocation of a system implemented in dedicated hardware and the same system implemented in a soft radio. While both approaches strive for compactness and power efficiency, a soft radio design requires additional flexibility and reusability, which impacts the partitioning of a process into its hardware components. This resource allocation is also closely related to the data flow properties of the system since data has to be processed with minimal delays and overheads.

Another important design issue is the smooth reconfiguration of the radio, including the ability of run-time reconfiguration and over-the-air updates. Smooth reconfiguration is necessary to minimize delays and make the system robust and easy to handle. The reconfigurable feature of a soft radio gives rise to some secondary design issues like reusability of hardware, scalability, and processing power. For mobile communication systems, power consumption is an important issue. Reusability of hardware supporting multiple communication standards is what makes a soft radio efficient in terms of silicon utilization.

A soft radio should be inherently scalable, both at the system level and at the hardware element level. Hardware scalability supports the easy addition of more computational hardware, ideally as easily as adding additional memory to a conventional processor. At the system level, the radio has to be conducive to *replication*, i.e., it should be easy to add channels to support more users. Finally, the hardware platform must provide high-rate computational processing.

Soft Radio Research and Commercialization

DARPA's Adaptive Computing Systems Project: Development of Commercial Off The Shelf (COTS) hardware components, design, programming, and runtime environments to enable an application to reach through to the hardware layer and directly manipulate the datapath-level architecture at runtime to optimize the application-level performance; exploration of new techniques that result in rapid realization of algorithm-specific hardware architectures on a low-cost COTS technology base. *http://www.darpa.mil/ito/research/acs/index.html.*

Virginia Tech: Research in developing algorithms and architectures for soft radios under the sponsorship of DARPA's GloMo I & II projects. Implementation of a multiuser receiver based on reconfigurable computing under GloMo I [2]. Development and implementation of a generic soft radio architecture for reconfigurable hardware under GloMo II. *http://www.mprg.ee.vt.edu/research/glomo/index.html.*

UC Berkeley: Work on Pleiades, part of the Adaptive Computing Systems project to achieve ultra-low power, high-performance multimedia computing through the reconfiguration of heterogeneous system modules. Achievement of high power efficiency by providing programmability at the right granularity and exploiting other energy reducing techniques, such as parallelism, pipelining, and dynamic voltage scaling. *http://bwrc.eecs.berkeley.edu/Research/Configurable_Architectures.*

Brigham Young University: Development of JHDL, a Java-based Hardware Description Language to facilitate hardware synthesis in reconfigurable processors. *http://jhdl.ee.byu.edu/docs/jhdlDocs.html.*

Chameleon Systems Inc.: Development of reconfigurable processor platform architecture to reduce tradeoff between performance and flexibility needed in wireless base stations.*http://www.chameleonsystems.com/newspg.htm.*

Morphic Inc.: Programmable hardware reconfigurable core using Dynamically Reconfigurable Logic (DRL) to replace multiple hardwired logic cores in wireless baseband processors. *http://www.morphics.com/html/tech.html.*

Quicksilver Technology Inc.: The Wireless Universal 'Ngine, or WunChip supports a variety of baseband algorithms that are downloaded from software. *http://www.quicksilvertech.com*

Sirius Inc.: Software-reconfigurable CDMAx test chip supports mobile station and base station configurations; can also be reconfigured as a GPS navigation baseband receiver and as a Satellite UMTS transceiver core.*http://www.siriuscomm.com.*

While most researchers agree on the usefulness of the soft/software radio and most of its desirable features, there is a lack of general design methodology applicable to reconfigurable systems. Most designs are based on ad hoc

approaches which are only appropriate for the problem at hand. There is a lack of a formal design methodology needed for the systematic realization of soft radios. Though much work is being done in the software and network layers of the soft radio, there has been little progress in the development of a generic hardware architecture for a soft radio.

In this paper we develop a formal design philosophy and methodology for a soft radio architecture. We show that the architecture can handle complex data processing with efficient resource allocation, while maintaining hardware reusability, flexibility, and scalability. The design methodology is illustrated with an example that has been implemented in hardware.

3 The Layered Radio Architecture

3.1 Overview of the Layered Radio Architecture

The approach to designing a reconfigurable radio using *hardware paging* is formalized in the layered radio architecture. Hardware paging refers to hardware modules being paged in and out of the system in a manner similar to software paging performed with the use of virtual memory. The layered architecture leverages on stream-based processing where a common bus is used for data as well as programming information. A few bits within the bus are reserved for the header, which indicates the nature of the stream packet. The use of the stream-based processing simplifies the interface between modules, making it easy to replace one module with another or add additional modules. The use of a reconfigurable hardware platform enables hardware paging. The architecture can handle complex data processing with efficient resource allocation, while maintaining hardware reusability, flexibility, and scalability.

The functionality of the radio is divided into layers, where each layer attaches/modifies the header and passes the information to the next layer. Once the processing is complete, the information is sent back through the layers in the same way. The layered radio architecture defines three layers (Figure 2), the Soft Radio Interface (SRI) layer, the configuration layer, and the processing layer. The application software residing above all these layers consists of high-level graphics and software that interface the radio with the user. The three layers of the soft radio are implemented using stream-based processing. The SRI layer is responsible for interfacing the radio hardware with the external world, coordinating the various sources of information coming in and going out of the radio, prioritizing the requests made to the radio, and packetizing the incoming information before sending it to the lower layers. The information coming into the radio can be either data, or programming information requesting a new setup of the radio, or a modification to the existing setup. The setup of a new system is handled as follows.

The SRI layer contains the system-level description of various radio configurations in its local memory. Each system-level description contains a list of all the algorithms in the appropriate order needed to implement the sys-

tem. It is important to note that the SRI layer does not contain the bits needed to configure the hardware. Rather, it contains the codes for the algorithms that are to be used and their sequence of interconnection. The configuration layer has in its memory the actual bits needed to configure the processing layer hardware. The configuration layer extracts the configuration bits from its memory based on the programming packets sent by the SRI layer. The processing layer consists of a series of reconfigurable modules called processing modules that perform the actual operations on the data to implement the functionality of the radio (Figure 3). Each processing module has a static and a reconfigurable section, where the static section configures the reconfigurable section based on the programming packets sent by the configuration layer. Once the radio is configured, data is packetized by the SRI layer and sent to the lower layers using the same bus used for the programming packets. After completion of processing, data is sent back through the configuration layer to the SRI layer, which in turn de-packetizes the data and outputs it to the host PC.

3.2 Use of Stream-Based Processing in the Layered Architecture

The soft radio architecture being developed at Virginia Tech is based on a concept called *stream-based* processing. A stream is a packet of known length containing either programming (configuration) information or the data to be processed. Stream-based processing provides a means to exploit the processing power attainable through deep pipelining, while still maintaining some degree of flexibility [6]. The algorithm to be implemented is first represented as a *data flow graph*. The data flow graph is then decomposed into smaller computational primitives called *processing modules*. Each processing module performs a unique subset of the overall processing on the data and then passes the data and control information to the next module. An analogy can be drawn to an assembly line process where each module performs a specific task as the component moves forward in the assembly line, as shown in Figure 1.

4 Implementation of the Layered Approach

4.1 Soft Radio Interface Layer

The Soft Radio Interface (SRI) layer receives the incoming digitized RF data and control information from the host PC that contains the desired configuration and settings of the radio. The SRI layer transmits processed data to the host PC, performs initialization of the system on power-up and performs the first step in configuring the radio. This layer has the system level description of the desired radio configuration in its local memory, which contains the algorithm codes for each processing module needed to implement the system. This system level description is sent to the configuration layer, which in turn configures the processing layer modules.

The incoming data to the SRI layer is either from the A/D converter (which always provides data) or control and status requests from the host PC. Once the user decides on the algorithms to be used, the SRI layer inserts the corresponding codes and forwards it to the configuration layer. Data that follows will be processed using these preset algorithms until the user decides to change the configuration of the system. Control packets containing over-the-air updates are used to change the configuration of the library. The requests from the PC (control information), as well as data from the A/D, is buffered and prioritized by the SRI layer and sent to the configuration layer.

The stream packets received by the SRI layer from the configuration layer have more information in them. The configuration layer sends the processed data back to the SRI layer along with other status and error messages. First, the SRI layer checks to see if the data contained in the packet is valid. If it is valid and the packet contains data, it is sent to the host PC, along with error messages if any. When the SRI layer sends program or control information that was requested by the host PC to the configuration layer, the packets from the configuration layer will contain acknowledgments that indicate if the operation was successful. These messages can either be sent back to the host PC or the SRI layer can re-transmit the control packets.

4.2 Configuration Layer

The information from the SRI layer indicates whether the packet is a control or data packet. If it is the former, the packet contains the algorithm codes of the desired configuration. For example, the control packet may request a DQPSK demodulator followed by a Viterbi decoder. The appropriate configuration code is extracted from the local memory and sent to the processing layer with the address/addresses of the processing layer module/modules. The local memory contains a table that has the addresses of the stored configuration code for each operation. The configuration layer also maintains a status list of the processing layer modules. When the configuration layer receives a data packet, the configuration layer headers are attached to the data and sent to the processing layer. Data coming in from the processing layer is sent back to the SRI layer for appropriate action.

4.3 Processing Layer

The processing layer forms the core of the soft radio that performs the actual processing of data. This layer is responsible for processing data sent from the configuration layer and sending the processed data back to the configuration layer. It consists of a set of linearly connected processing modules, as shown in Figure 3. Each processing module has the capability of reconfiguring itself to perform the desired function without disrupting synchronization in the main pipeline. The main flow of data is in the forward direction from the first to last module, as shown in Figure 3. Each processing element maintains the continuity of this stream pipeline at all times. Each processing element has both, a static and a reconfigurable section. The static sections

have the capability of configuring the reconfigurable section using the control packets sent by the configuration layer. Data can thus either bypass each processing element entirely or be routed through the reconfigurable section for processing. The structure and operation of the processing element is described in the next section.

Many communication circuits have feedback loops. To accommodate such systems, the stream pipeline is designed to be bi-directional in the processing layer. The input/output ports of the processing element can either be implemented as bi-directional ports or can have a separate bus in the reverse direction. The reverse slots/reverse buses are used to implement feedback loops. If the ports are bi-directional, the main clock cycle is divided into forward and reverse slots, and the directions of the interconnections are reversed in the reverse slots. If a separate reverse bus is used, then with every clock cycle, a module both accepts and outputs a forward and reverse packet to the neighboring module. Reverse packets are not needed in the absence of feedback loops.

When valid data enters an appropriate module, it is processed and sent to the next module. The address field is updated to indicate the next module that needs to act on the data. Similarly, if there is control information in the packet for the module, the corresponding code is executed to change the configuration of the reconfigurable section. If there is any latency while the module is being reconfigured, the static portion of the module buffers the subsequent incoming data and sends out idle packets. However, if it gets control packets intended for modules further down the chain, then those packets are bypassed and sent to the next module. Thus the continuity of the stream is maintained at all times.

Data in the stream packet is sent to the processing pipeline. At the end of the processing pipeline, the packet is reconstructed, and error or status messages if any are added to the header. When the processing is completed, data is sent from the output module back to the configuration layer and finally to the SRI layer that delivers the output data. If the packet contains program information, the module checks to see if the address on the header matches the module address. If so, the code present in the body of the packet is executed in the configuration pipeline, thereby changing the configuration of the processing pipeline. It is also possible to have special error packets to handle more serious errors. If an error occurs in the execution of the code, the module sends a signaling packet indicating the location and nature of the error to the configuration layer. The address field is set to a special value so that none of the succeeding processing modules intercept the packet containing the error message.

4.3.1 Structure of the Processing Element

The functional description of each processing module is shown in Figure 4. Running on each processing element are three sets of pipelines and a state machine that interprets how the packet is channeled through the pipelines. The processing pipeline processes the data, i.e., performs an operation on

the data as part of the demodulation process. The configuration pipeline controls the hardware configuration of the processing pipeline. The ability of the configuration pipeline to execute programming information makes it possible to modify the low-level parameters and functionality through high-level software. The bypass pipeline allows the packet to bypass the module if the packet is not intended for the module. The bypass pipeline is present to ensure that the module does not corrupt data it is not supposed to act on. It is important that each of these pipelines have the same amount of delay, so that the packets are synchronized with the main clock cycle. At the end of the pipeline, the stream packet is reconstructed with the updated header and routed to the next module.

Figure 4 gives a functional description of the processing module, which varies slightly when implemented in hardware. The static sections are identical in all processing modules. The static section of the processing module consists of a packet interpreter, configuration pipeline, and packet reconstructor. The packet interpreter is essentially a switch. By looking at the header bits, all control packets are sent to the configuration pipeline and all data and signaling packets are sent to the processing pipeline. The function of each of these packets is described in Section 3.2. The configuration pipeline accepts all the control packets and has the capability to configure the processing pipeline accordingly. The modules maintain the configuration to which they are set and act accordingly on valid data until the configuration is changed. The packet reconstructor accepts data from all the pipelines, buffers them if needed, and sends them out to the next module. The processing pipeline is located in the reconfigurable section of the processing module. The reconfigurable section of the processing module also contains a data interpreter and data reconstructor that takes care of the byte ordering for the data going in and coming out of the processing pipeline.

When a stream packet enters a processing module, the module interprets the packet and performs the necessary action. The processing element examines the contents of a packet only if the valid bit is set. The valid bit is introduced to maintain synchronization between the modules. Every clock cycle, each processing element accepts a packet and sends out a packet. However, the module acts on the packet only if the valid bit is set. Similarly, a module sends out an invalid packet instead of inserting wait states.

4.4 Implementation of an Adaptive Single-User CDMA Receiver

At Virginia Tech, the layered architecture was used to develop a single-user CDMA receiver with an LMS equalizer and differential BPSK demodulation. The receiver was designed using adaptive filtering that performs both despreading and equalization. The parameters which can be varied include the equalizer algorithm, filter length, spreading code and step size. The receiver was implemented on the Giga OPS, G900 Board by Spectrum, that holds the XC4028EX-series of Xilinx FPGAs, operating at a modest 1.25 MHz. The RF front-end filtered, amplified, and mixed the received signal down to an

IF of 68 MHz. The IF was fed into the Harris digital downconverter on the Sigtek evaluation board, ST-114, where it was sampled by the on-board A/D converter. The signal was then digitally downconverted to baseband, and the chip rate was decimated to 1 MHz. A picture of the receiver is shown in Figure 5 which shows the transmitter, the ST-114 Sigtek evaluation board, and the G900 Board by Spectrum. The output of the ST-114 was the complex envelope of the received signal. It consisted of two sixteen-bit buses, which represent the in-phase and quadrature channels.

Due to limitations in the hardware, the SRI and configuration layers were combined into one unit called the input module. The processing modules consisted of the adaptive filter and decision module, and the acquisition and tracking module. The main clock was divided into eight slots with four forward slots, two reverse slots and two guard slots. The guard slots were present to prevent two modules from outputting data to the same bus simultaneously, when the direction of data flow was reversed. The forward slots were used to transmit data and programming packets. The reverse packets were used for feedback including synchronization and timing information. The processing modules were implemented using the structure shown in Figure 4. The length of the adaptive filter, convergence factor of the LMS algorithm, and the user spreading code could be varied in the receiver system.

On power up, the address bits of all the processing modules were set to 0. Subsequently address initializations were performed using programming packets sent by the input module. When the first processing module encountered a valid programming packet, the module assigned itself the new address and set the invalid bit in the stream before sending the packet to the next module. This ensured the other modules did not get the same address. When the next programming packet with a new address was sent by the input module, the first module does not act on it, since its address is now non-zero. The next un-initialized module assigns itself the new address and the process continues until all the modules have a unque non-zero address. The next set of programming packets perform other initilaizations including setting the user spreading code, filter length, etc. Data is sent through the system after completion of all programming.

The implementation on hardware consumed about 300,000 gates. The implementation also indicated the need for more sophisticated hardware with run-time reconfiguration capabilities for successfully implementing the layered radio architecture. The hardware requirements are further elaborated in Section 5.

4.5 Advantages and Tradeoffs in the Layered Architecture

A number of wireless standards have emerged in the recent years. However, when third generation (3G) systems are launched, they will have to co-exist with second and even first generation wireless systems [7]. The proposed 3G systems differ from existing standards in their modulation formats, channel coding, multiple access, etc. There is a need for reconfigurable, multiband,

multimode terminals and base stations that can accommodate mobile users across different standards. Reconfigurability thus becomes extremely important in soft radios. It is not adequate that the radio be configured only in software, but the hardware itself has to be reconfigurable to be able to implement a wide variety of standards. The layered radio architecture defines the methodology to do this using hardware paging, without making any assumptions on the actual algorithms used in the standards. The layered radio arhcitecture can thus be used as a general methodology to design any multimode radios.

The layered architecture provides the framework for building a flexible soft radio at the expense of the overhead for packetizing data. It is possible to implement radically different radios on the same hardware platform through the use of this architecture. Additionally, the architecture is conducive to the implementation of adaptive modulation schemes, achieved by reconfiguring the hardware in real-time.

One of the main advantages of the architecture is its excellent hardware reusability. For example, core units like BPSK and QPSK demodulators can be used in a wide variety of standards when built in hardware with the right parameters. Thus it is possible to build libraries of hardware functions much like building libraries of software functions.

The layered architecture has good data flow properties and a simple interface between the processing layer modules, which make it scalable at the system level as well as the hardware element level. The use of stream-based processing ensures that each processing module has one input and output bus that is common for control information and data bits. This simplifies the interface between two processing modules and makes it easy to add additional hardware modules or integrate designs from different engineers.

Over-the-air updates can be performed by changing the contents of the libraries of the SRI configuration layers, which can be accomplished by two different approaches. The actual bit files stored in the libraries can be transmitted to the radio and each layer modifies its library based on the programming information accompanying it. This approach, however, requires the transmitter to have prior information about the specifics of the hardware, which could be vendor-dependent. This approach, thus, may not be the most commercially viable solution. Alternatively, some standardized high-level programming language can be used to transmit the code to the radio. The SRI layer then compiles the code to generate the bit files needed for that radio. The SRI layer modifies its own library contents using the newly generated bit files. The SRI layer also packetizes the bit files corresponding to the configuration layer and assembles them into appropriate programming packets. The second approach requires a hardware compiler in the SRI layer.

5 Insight into Hardware Development

The layered radio architecture makes certain assumptions about the processing layer in addition to the hardware being reconfigurable. The configuration layer should be able to reconfigure the processing modules, independent

of other modules. This inherently assumes partial reconfigurability of the hardware, along with a very high reconfiguration speed. Virginia Tech has been involved in developing a FPGA specifically suited for flexible, high-throughput, low-power computations, called the *Stallion* that is based on wormhole reconfigurable computing [4]. The architecture of the Stallion is shown in Figure 6. It consists of six bi-directional dataports, two 8x4 meshes of interconnected functional units, a crossbar and four hardware multipliers. The functional units are programmed to process data while the crossbar aids in routing data through Stallion.

The essence of wormhole run-time reconfiguration is that independent, self-steering streams of programming information and operand data interact within the architecture to perform the computational problem at hand. The stream is self-steering, and as it propagates through the system, configuration information is stripped from the front of the header and is used to program the unit at the head of the stream. Since the streams independently guide themselves through the system using the information contained in the stream header, the configuration process is inherently distributed. Multiple independent streams can wind their way through the chip simultaneously. Part of the system can be used to process data operands while any set of neighboring units are accepting configuration data from the header of one or more streams, thus allowing overlap of the configuration and processing operations, as well as partial reconfigurability. The use of Stallion for the processing modules makes it possible to implement the full functionality of the radio, which cannot be achieved with the use of commercial FPGAs. With the use of Stallion, the processing modules can be partially configured as well as configured during run-time as required by the arhcitecture

6 Conclusions

The layered architecture presents an open yet formal and unified structure for implementing soft radios on reconfigurable platforms. The layered approach lends itself easily to standardization of soft radio systems and also allows code reuse. The processing modules and header information can also be easily standardized, allowing support from a variety of vendors. One of the main advantages of the structure of the processing modules is that the interface between two modules is very simple. Each module is able to accept a stream packet and send out a stream packet in the forward and reverse directions in each cycle. The layered architecture is scalable and completely flexible.

The layered architecture is suited for today's FPGAs that support partial reconfiguration and for tomorrow's configurable computing platforms. Virginia Tech's *Stallion* is one such architecture that can support the layered architecture in an efficient manner. Flexibility, of course, is gained at the cost of additional overhead in dealing with packetized information. Current research at Virginia Tech focuses on the development and testing of a soft radio based on the layered architecture and building a library of soft radio modules.

In this paper we have presented a scheme whereby the processing modules

are connected in a sequential manner. Connecting the modules in a sequential manner is a first step in designing soft radios, but a more sophisticated and powerful approach is to network these modules in the processing layer. For example, creating a mesh of processing modules with 4-connectivity, can increase the processing power tremendously. This of course opens up a whole new area of research where the stream packets no longer travel in just one (or two) directions. This design approach would have to consider routing aspects, packet collision and prevention, and packet ordering, which are the same issues encountered in packet radio communication systems. However, this change can be viewed as just an extension of the layered architecture. A new layer, a *network layer*, can be inserted between the configuration layer and the processing layer that takes care of the packet routing and collating of information obtained from the processing layer. Other aspects of the architecture can remain unchanged.

Acknowledgments

Support to this project was provided by the Defense Advanced Research Projects Agency (DARPA) GloMo program, the Navy Collaborative Integrated Information Technology Initiative program and Conexant Systems, Inc.

References

[1] Peter M. Athanas, J. H. Reed and W. H. Tranter, *A prototype software radio based on configurable computing*, Advancing Microelectronics, pp. 34-39, 1998.

[2] Peter M. Athanas, I. Howitt, T. Rappaport, J. H. Reed and B. Woerner, *A high capacity adaptive wireless receiver implemented with a reconfigurable computer architecture*, ARPA GloMo Principle Investigators Conference, San Diego, CA, November 1995.

[3] Mike Butler, James Providakes, Gary Blythe, *The layered radio*, MILCOM 98.

[4] Ray Bittner, *Wormhole run-time reconfiguration: Conceptualization and VLSI design of a high performance computing system*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, Virginia Tech, 1997.

[5] J. K. Cavers and M. Liao, *Adaptive compensation for imbalance and adaptive losses in direct conversion tranceivers*, Proceedings of the 41st IEEE VTC, pp. 578-83, 1991.

[6] DeCegama, Angel L., *Parallel processing architectures and VLSI Hardware*, Englewood Cliffs, NJ, Prentice Hall, vol. 1, pp. 157-160.

[7] Pentti Leppanen, Jaakko Reinila, Asko Nykanen, Visa Tapio et. al., *Software radio-An alternative for the future in wireless personal and multimedia communications*, IEEE International Conference on Personal Wireless Communications, pp. 364-368, February 1999.

[8] Joseph Mitola, III, *Software radio architecture: A mathematical perspective*, IEEE Journal on Selected Areas in Communications, vol. 17, no. 4, April 1999.

[9] Joe Mitola, *The software radio architecture*, IEEE Communication Magazine, pp. 26-38, May 1995.

[10] Srikathyayani Srikanteswara, Peter M. Athanas, Jeffrey H. Reed and William H. Tranter, *Configurable computing for communication systems*, Proceedings of the Wireless Communications Conference, IMAPS, pp. 180-185, 1998.

[11] Hiroshi Tsurumi and Yasuo Suzuki, *Broadband RF stage architecture for software-defined radio in handheld terminal applications*, IEEE Communications Magazine, pp. 90-95, February 1999.

7 Figures

Figure 1: Stream-based processing

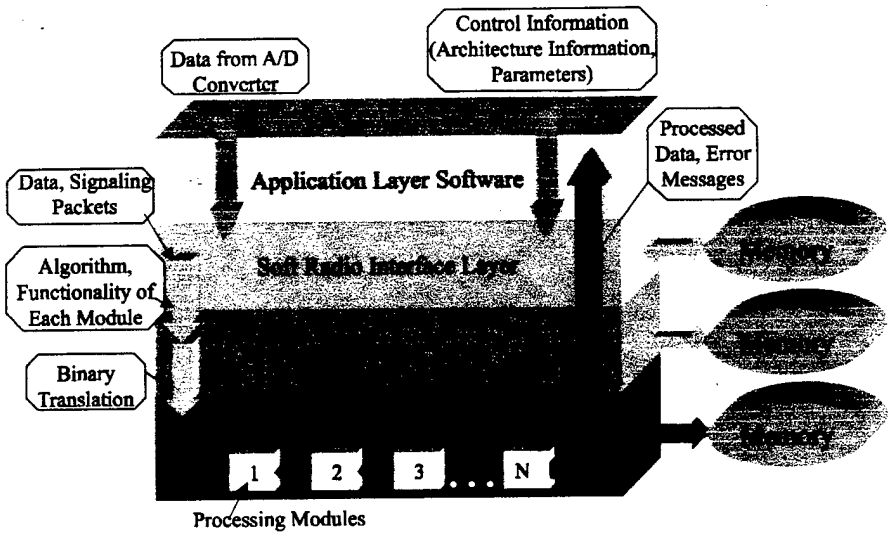


Figure 2: Layered architecture

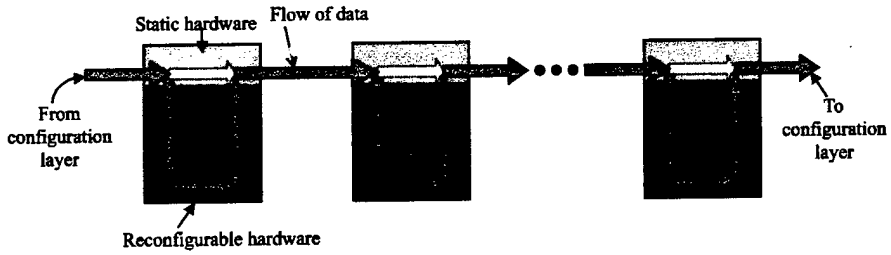


Figure 3: Processing elements in the processing layer

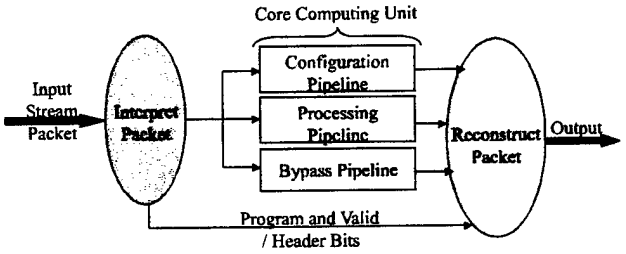


Figure 4: Functional description of a processing element

Figure 5: Single-user CDMA receiver with adaptive filtering

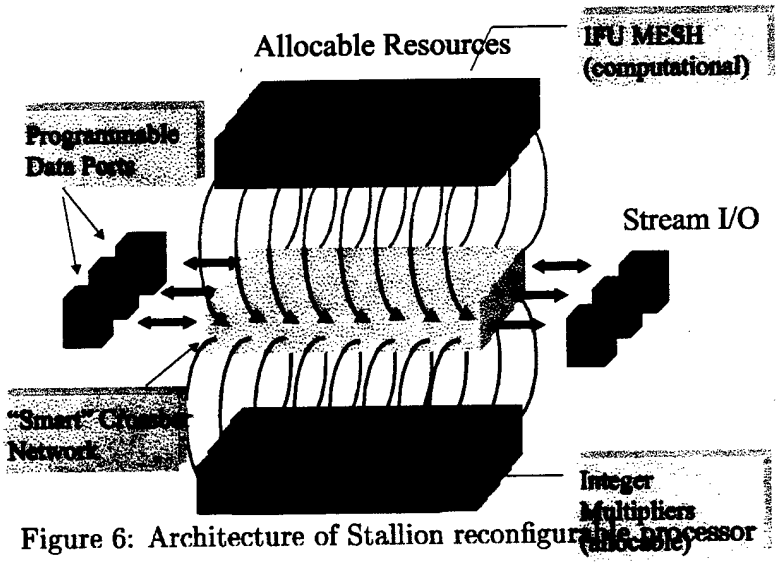


Figure 6: Architecture of Stallion reconfigurable processor

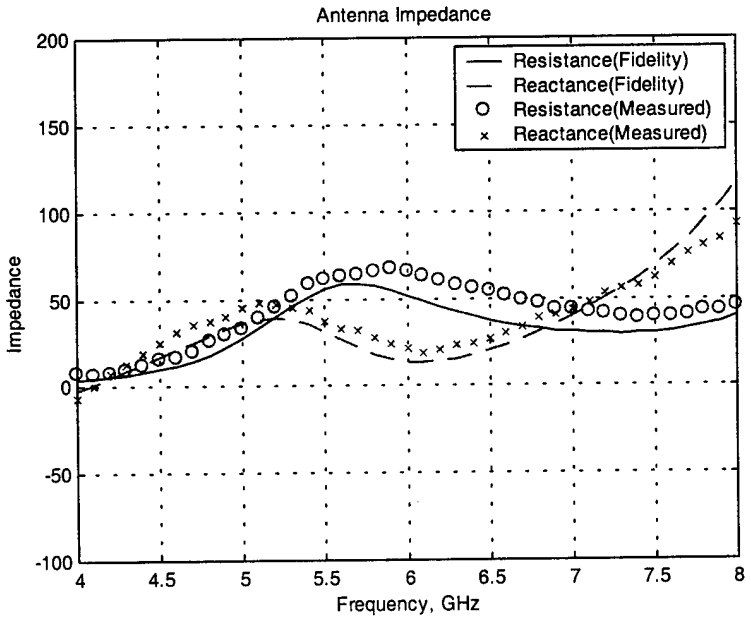
2.2 Recommendation for Radiating Elements, and collaboration with Navy partners (Warren. Stutzman, Tasks 1.2.2, and 1.5.1) CDRL Data Item 009, 0029

1. Foursquare Element Research

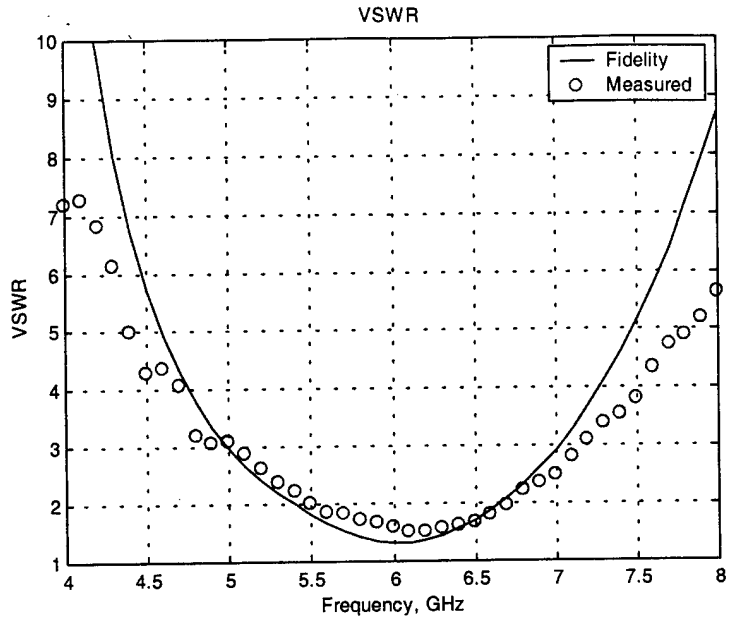
The Virginia Tech Antenna Group modeled the Foursquare element using the FDTD (Finite Difference Time Domain) commercial code Fidelity. The computer model of the radiating element is very similar to a hardware model constructed by the Antenna Group except for differences in the substrate thickness and the ground plane. The values calculated using Fidelity are plotted in Fig. 1-1. The results obtained for the Foursquare model using the Fidelity code are very similar to the real element manufactured and tested by the Virginia Tech Antenna Group. Slight differences between modeled and measured results can be attributed to differences in substrate thickness and ground plane area. The model used 30 mils substrate thickness instead of the 28 mils substrate thickness with the hardware in order to avoid using a large number of cells in the computer model. Also, the computer model used an infinite ground plane instead of a finite ground plane as used in the hardware element. However, the length of the ground plane in the hardware element is almost ten times that of the radiating element, approximating an infinite size condition.

Methods of increasing the bandwidth of the Foursquare element were investigated using Fidelity by changing the substrate thickness. The ground plane spacing was fixed at t_d = 0.25". The substrate thickness was varied as follows: t_s = 30 mils, 10.5 mils and 0 mils. Simulation results shown in Fig.1-2 demonstrate that a thinner substrate gives better performance. The bandwidth of the element with t_s = 30 mils is 20% for 2:1 VSWR and the bandwidth for t_s = 10.5 mils is increased to 34.4%. Without any substrate (t_s = 0), a bandwidth of 51% was achieved.

Future work will focus on investigating the effect of ground plane size on the antenna performance. In addition, we will explore ways to reduce the antenna size while keeping on the same antenna performance.

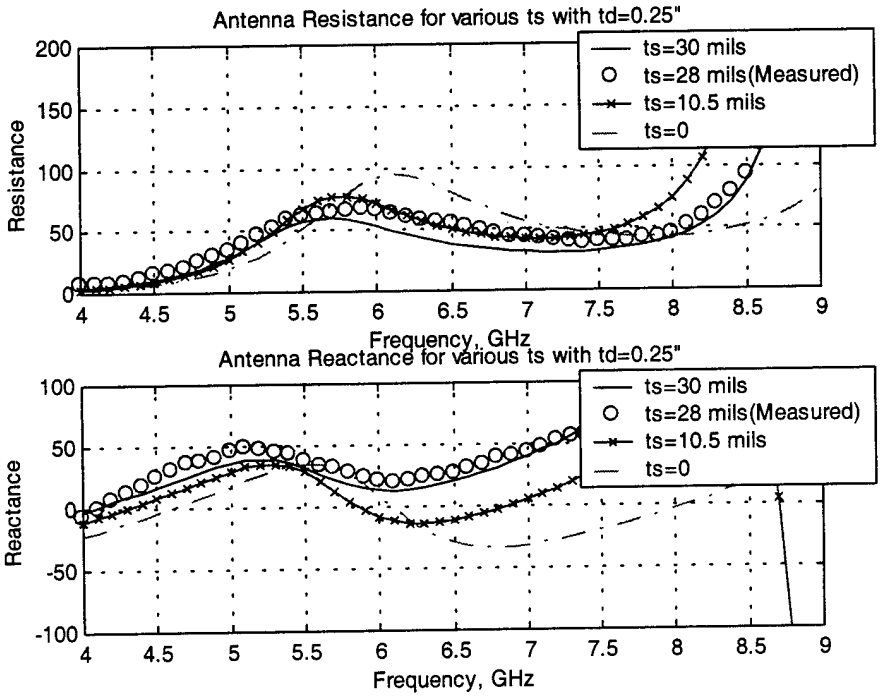


(a) Antenna input impedance

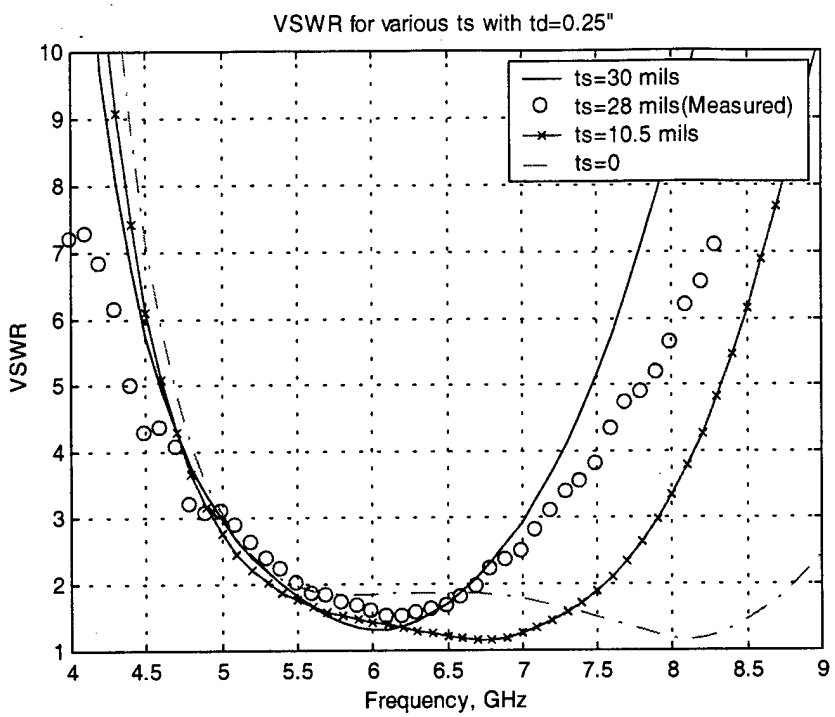


(b) VSWR

Figure 1-1 Comparison of measured and simulation results for the impedance and VSWR of a Foursquare element.



(a) Antenna input impedance



(b) VSWR

Figure 1-2 Comparison of measured and simulation results for the impedance and VSWR of a Foursquare element for various substrate thicknesses.

2. Foursquare Array Analysis

We investigated the sensitivity of the input impedance of a single Foursquare element to variations in height above the ground plane compared to that for an element in an array. To simulate the array effects, the Foursquare element was excited at the center of the two-by-two array configuration shown in Fig. 2-1. The input impedance was calculated for element heights above the ground plane of h = 0.14", 0.18", 0.21", and 0.28"; see Fig. 2-2. The center frequency shifts from 7.6 GHz to 6.2 GHz for increasing height above the ground plane. Similar trends were observed in the single Foursquare input impedance.

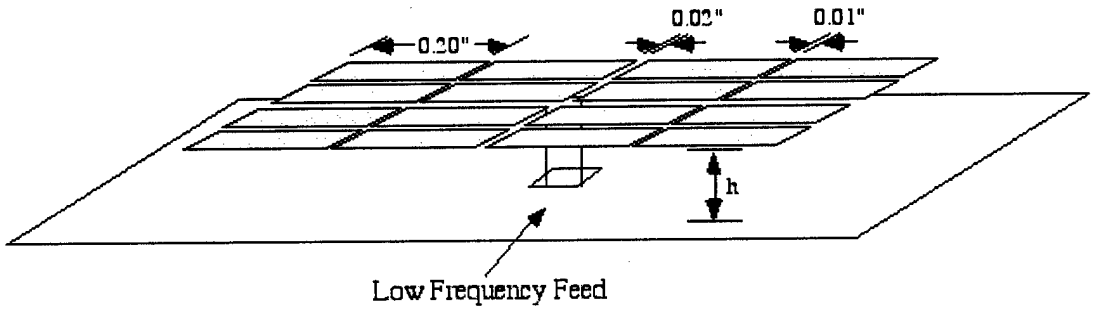


Figure 2-1 Geometry of a four-element array of Foursquare antennas in a two-by-two configuration.

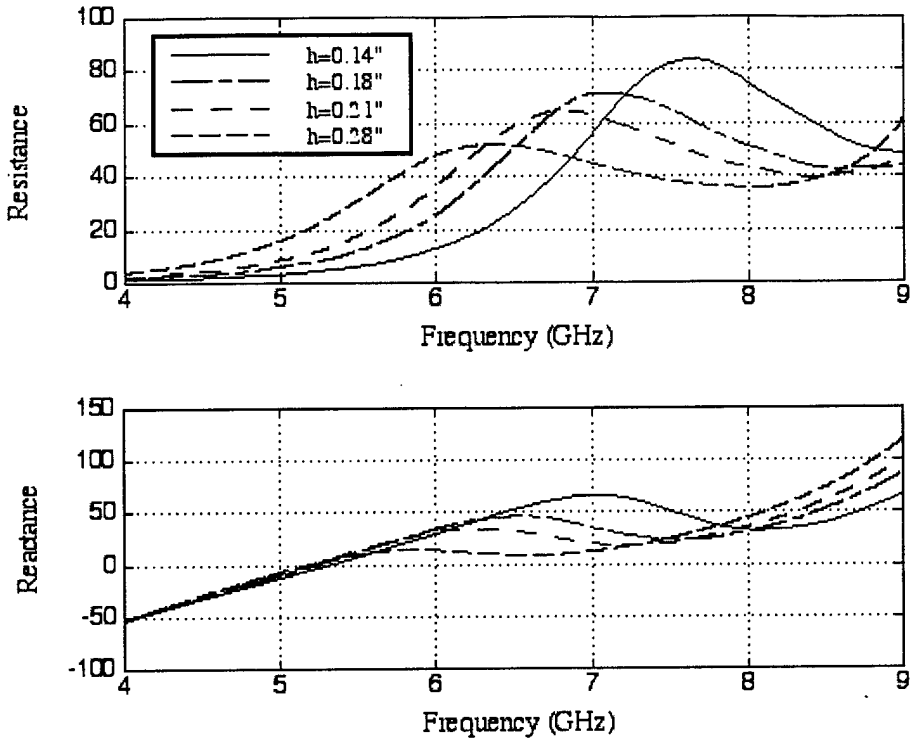


Figure 2-2 Calculated real and reactive input impedance for varying heights above the ground plane for the four-element array of Foursquare antennas in the two-by-two configuration of Fig. 2-1.

3. Equipment Acquisitions

Task 1.2.2 includes money for a near field scanner and associated equipment to measure wideband antennas, as well as funds for software to simulate wideband antennas. All software was acquired and put in use during the first three quarters. In the fourth quarter the hardware was received. The antenna measurement system consists of a Hewlett Packard vector network analyzer and an Antcom near field scanner. The network analyzer was received and tested in August. The near field scanner was received in October and is undergoing final testing. All equipment is now in place. Final system evaluations will begin soon, followed by testing of the wideband antennas.

2.3 Description of a trial application (Richard Nance- Task 1.3.1) CDRL Data Item 0013
Our work over the past two months has been to get our Distributed Systems Laboratory in an operational status. That has proved to be more difficult than we had hoped (but about the same as expected). This laboratory will enable us to be a major player in the Virtual Operations Networks project in Year 2.As for Year 1 (Software Quality Measurement), we have examined and reviewed four candidate frameworks: (1) Practical Software Measurement (PSM), (2) the SEI Capability Maturity Model, (3) the Objectives/Principles/Attributes (OPA) framework, and (4) an approach based on subject matter experts applying aspects from all of the others. We believe this last approach to offer some major advantages, and we are intending to suggest this in the future system applications.

We are in the process of setting up a meeting with Dr. William Farr and Ms. Sherry Barker of NSWCDD to discuss our approach and the potential for applying the work to a Navy project.

2.4 Description of Prototype Virtual School (John Carroll- Tasks 1.3.2,1.4.1, 1.6.1) CDRL Data Item 0015, 0016, and 0031.

Our work during this past year has focused on usability engineering and design guidelines for developing interactive virtual environment (VE) applications. Specifically, we have focused on designing navigation in a Responsive Workbench for an application, called Dragon, developed at the Naval Research Laboratory in Washington DC. We are addressing the applicability of established usability engineering methods in conjunction with development of new usability engineering methods. Our goal is to provide a methodology — or set of methodologies — to ensure usable and useful VE interfaces. Our work has produced a high-level methodology that is a logical progression of techniques. This work has been summarized in CDRL #0025 which is being submitted separately.

The Center for Human-Computer Interaction has two areas of responsibility in the first year of the NAVCIITI program: The first area is in design rationale, an intersection between human-computer interaction (HCI) and software engineering. We are investigating the proactive use of design rationale in software development, specifically focusing on the management of usability rationale in a scenario-based design process through use of claims analysis, and including refinement and dissemination of such methods for analyzing usability and domain object tradeoffs in Naval software designs, prototypes, and systems. Our task in this area is to prepare a textbook for the methodology, and to support Naval software and system development.

The second area of responsibility for the Center for Human-Computer Interaction is in computer-supported collaborative work (CSCW). We are investigating the development and application of collaborative multimedia conferencing software for education and other groupwork activities. We are extending our virtual school software framework integrating shared notebook, whiteboard, chat, visualizations, simulations, video conferencing, etc.

We have continued to make good progress on our technical tasks. An upper level textbook on the management of usability rationale in a scenario-based design process through use of claims analysis is complete. We have begun work on a lower-level textbook, which presents scenario-based system development as part of the undergraduate computer science curriculum in software engineering and human-computer interaction. Addison-Wesley, Morgan-Kaufman, MIT Press, Prentice-Hall, John Wiley and Lawrence Erlbaum Associates have expressed interested in publishing this text, based on reviews of a chapter-by-chapter outline. Chapters 1-6 and 9 are now complete, and are being used in as text materials in CS3724 "Introduction to Human-Computer Interaction" in the Fall 1999 semester. Fourteen chapters are planned. Based on this material, we proposed, and have now been invited to provide, a full-day tutorial on scenario-based design for the 2000, ACM CHI Conference, the top international conference in human-computer interaction.

We have also made progress on several fronts in developing a system to support collaborative learning and planning interactions. We are focusing on developing and evaluating the Virtual School, a Java-based networked learning environment, emphasizing support for the coordination of synchronous and asynchronous collaboration. The Virtual School integrates communication tools such as video conferencing, shared whiteboards, chat, and email. It also includes a shared notebook that supports collaborative planning, note taking, experimentation, data analysis, and report writing. Currently we are developing several new collaborative page types for the shared notebook, investigating off-the-shelf component reuse, experimenting with a set of synchronous and asynchronous awareness tools, and developing a comprehensive evaluation methodology to assess the usability and usefulness of such environments. To test new tool development and

further explore the evaluation framework, we are continuing to implement the system in a number of new contexts. Investigating a range of different cooperative work types under varying synchronous and asynchronous modes is allowing us to expanding our understanding of how the tools are used in real contexts and how to evaluate them under these conditions. For example, in one new environment the tools are being used for teaching instead of peer-to-peer collaboration, and in another case we are analyzing asynchronous only mentoring.

One focus of this work in the past six months has been the development of a multi-faceted evaluation methodology for complex, distributed groupware activities. We are continuing to develop methods of large-scale usage data collection over the Internet. Evaluating distributed systems call for innovative methods that do not require an evaluator to be physically present at the user's work site. We are continuing to refine our tool set composed of three interrelated processes: (1) event capturing, (2) activity filters, and (3) integrated event scripts (collated activities from multiple data across a range of contexts). This form of data logging provides multiple representations of user and system behavior that translates measures of performance into more meaningful characterizations of human-computer interaction. We have begun to combine this usage data into integrated event scripts. These scripts are multi-threaded documents that characterize group activities collected from different connected sites. A range of data collected from the different sites are integrated into the scripts. We are specifically working on developing procedures that reduce the time needed to analyze data versus the duration of data collection sequences. We are trying to develop heuristic methods that maximize cost-benefit in the analysis to produce optimal data collection and analysis procedures. Part of this work is developing a classification scheme to map different problem types to different data collection methods. This will allow us to pinpoint how different methods find critical, typical, and problematic tasks.

A second focus has been the development of an alternative environment for place-based synchronous groupwork. The new environment is based on the same collaborative infrastructure as the Virtual School, and supports end-user construction of hierarchies of shared spaces. Persistent shared objects can be created within and moved between these spaces. Users can navigate from place to place using a map-based interface and can interact synchronously via text chat, whiteboards, and slide shows. They can interact asynchronously by creating and manipulating persistent objects, including message boards, shared notebooks, and Java applets. We are exploring the extent to which this interactive Java-based environment can be made more widely accessible and provide richer interactions than traditional place-based tools such as purely text-based multi-user domains (MUDs).

We have begun to interview potential clients and users of this alternative environment for place-based synchronous groupwork. This information is being used as an initial user profile, requirements analysis, and task analysis. Several groups have been targeted for data collection, and we are using the data for a side-by-side comparison to determine group appropriateness as test cases. Several significant enhancements have been made to the system architecture. Support has been added for allowing users to extend the environment's capabilities by developing new collaborative components. Current work includes ongoing development of flexible security mechanisms for places and objects in the environment.

Publications currently submitted and in review

Carroll, J.M., Isenhour, P.L., Rosson, M.B., Van Metre, C., Schaefer, W.A., & Ganoe, C.H. MOOsburg: Supplementing a real community with a virtual community. Submitted to International Network Conference 2000 (3-6 July, Plymouth, United Kingdom).

Chin, G. & Carroll, J.M. 1998. Articulating collaboration in a learning environment. Submitted to *Behaviour and Information Technology*.

Dunlap, D. et al. 2000. Teacher collaboration in networked communities. *Educational Technology & Society*, Special Issue on "On-Line Collaborative Learning Environments"
Go, K. & Carroll, J.M. 1998. Blind men and an elephant: Views of scenario-based system design. Submitted to *ACM interactions*
Helms, J., Neale, D.C. & Carroll, J.M. Data logging: higher-level capture and multi-level abstraction of user activities. Submitted to *Annual Conference of the Human Factors and Ergonomics Society.*
Isenhour, P.L. et al. 2000. The Virtual School: An integrated collaborative environment for the classroom. *Educational Technology & Society*, Special Issue on "On-Line Collaborative Learning Environments"
Isenhour, P.L., Rosson, M.B. & Carroll, J.M. Supporting asynchronous collaboration and late joining in Java groupware. Submitted to *Interacting with Computers*, Special Issue on Web Software.
Neale, D.C., Dunlap, D.R., Isenhour, P.L. & Carroll, J.M. Collaborative critical incident development. Submitted to *Annual Conference of the Human Factors and Ergonomics Society.*
Submissions recently accepted for publication/presentation
Carroll, J.M. *HCI in the New Millennium.* Edited book to be published by Addison-Wesley/ACM Books, contract signed in August.
Carroll, J.M. & Rosson, M.B. Scenario-Based Usability Engineering. Tutorial for ACM CHI'00 Conference: Human Factors in computing Systems.
Gibson, S., Neale, D.C., Van Metre, C.A. & Carroll, J.M. Mentoring in a school environment. Accepted for publication/presentation *Third Conference on Computer-Supported Cooperative Learning.*
Neale, D.C. & Carroll, J.M. Multi-faceted evaluation for complex, distributed activities. Accepted for publication/presentation *Third Conference on Computer-Supported Cooperative Learning.*

2.5 Strategies to control ship motions in high seas and description of LAMP to Cave Interface (Ali Nayfeh- Task 1.4a.2) CDRL Data Item 0021, 0022
The objective of this task is to purchase, install, and operate the hardware and software that are needed to use state-of-the-art physical models of ships and cranes mounted on ships to develop a state-of-the-art Ship and Crane Simulator Test-Bed at the CAVE at Virginia Tech. To accomplish this objective, we are combining the existing VT CAVE hardware and software capabilities with the new hardware and developing the software capabilities required to support the Ship and Crane Simulator and raise simulations to a new level of reality.

For the ship response simulation, we are using the Large-Amplitude-Motions Programs (LAMPs) developed by the Science Applications International Corporation (SAIC), as being upgraded by SAIC and Virginia Tech under the Multidisciplinary University Research Initiative on Nonlinear Active Control of Dynamical Systems sponsored by ONR. LAMPs predict interactively the motion of and the flow around a ship advancing in moderate and severe sea conditions. They provide solutions in the time-domain and are not restricted to periodic motions. They can handle time-varying forward speeds and large-amplitude incident waves and large-amplitude ship motions where changes in the underwater surface of the ship are significant. The LAMP codes use potential theory to simulate the flow. They have varying degrees of complexity, including linear, weakly nonlinear, and fully nonlinear options. At the highest level, the free-surface boundary conditions are linearized about the actual incident-wave surface, and the body boundary conditions are satisfied on the instantaneous wetted surface. The wave amplitude may be the

same order as the draft of the ship, but must be an order of magnitude less than the wavelength. An integral equation for the velocity potential is obtained by using Green's theorem and time-dependent Green functions on the free surface and the wetted portion of the hull. This integral equation is solved by a panel method: the surfaces are divided into planar quadrilateral elements of constant potential strength and by collocation the integral equation is replaced by a finite system of linear algebraic equations. The resulting system of linear equations is solved at each time step; thus LAMP produces a time-domain solution that is not restricted to periodic motions. In order to account for viscous effects, which are dominant in the case of roll motion, an empirical state-of-the-art method based on experimental data is employed at the present. The resultant forces and moments are calculated at each time step.

We purchased a six-degree-of-freedom MOOG motion platform. Because it is large, we had to disassemble it and move it into suite 2400, 2000 Kraft Drive, Blacksburg, VA, and then reassemble it. An application programming interface (API), or library subroutine, was written for controlling the platform motion from an SGI Onyx using an RS-422 serial protocol. Computer programs using this API communicate new the desired motions to the platform indirectly and asynchronously visa inter-process shared memory. It is shared with a server process that writes the requested new motions to the RS-422 serial line. This design was needed in order to keep the RS-422 serial line active so the computer of the platform can function properly. SGI graphics programs do not run with a fixed cycle time so they are not capable of keeping the RS-422 serial line active and so a separate server process is needed. The shared memory design leads to an easy asynchronous control of the platform by any process. Hence, a separate real-time dynamics process can be run to control the motion of the platform. Using the developed API, we have written a computer program to have the platform move using data from LAMP simulations.

We started with the intention of using the marine module and other functionality of MultiGen-Paradigm's Vega. However, we dropped the use of because of its inability to synchronize in time the waves generated with Vega with the waves predicted with LAMP. Instead, we wrote a code with SGI Performer and the CAVE library. Since Vega is built on top of Performer, we are in effect programming with a lower level API in which we gain flexibility. We have developed such a computer program that displays ocean waves in the CAVE.

On Oct 4, 1999, at the Sixth Semi-Annual MURI meeting, we demonstrated a graphical ship model traveling through ocean waves, as predicted with LAMP. Moreover, we made the platform move according to the motions predicted with LAMP and generated the waves. In other words, the waves and motions are synchronized. On November of 1999, we mounted a truck driver's seat to serve as a crane operator seat on the platform. We also constructed a two arm-side console frame for mounting the crane operator control devices.

The standard tracking system for CAVEs and other immersive environments use magnetic fields for determining location and orientation. In an environment where there are large steel or iron objects, such as structural I-beams, the tracking system's data can be skewed due to the

object's interference with the magnetic field that's emanated by the tracking system. This skew can usually be accommodated if the metal object is not in motion. When the Moog motion platform, a 1700 pound steel structure, is placed in the floor of the CAVE, magnetic tracking becomes unusable. Not only does the steel's mass skew the field, but when the platform is in use the mass is in motion, and is placed in motion by use of electric motors. It became imperative that a tracking system should be found that did not depend on magnetic fields. Otherwise the motion platform could not be used in the CAVE.

After a search of the available products, the Intersense tracking system (http://www.isense.com/) emerged as the product that provided the highest speed and greatest accuracy. The Intersense

tracking technology incorporates a mixture of inertial and acoustic sensors which are unaffected by magnetic fields and metal objects.Intersense had not yet installed a unit in a CAVE, so the VT CAVE staff worked with Intersense to produce a viable system, which so far has met all expectations. The Intersense software is compatible with the original magnetic tracking system's software, so all original CAVE application and development packages were unaffected by installing the new system.

2.6 Specifications for Tactical Interface based on HCI principles (Deborah Hix- 1.4b.2) CDRL Data Item 0025

1. INTRODUCTION

Our work during this past year has focused on usability engineering and design guidelines for developing interactive virtual environment (VE) applications. Specifically, we have focused on designing navigation in a Responsive Workbench for an application, called Dragon, developed at the Naval Research Laboratory in Washington DC. We are addressing the applicability of established usability engineering methods in conjunction with development of new usability engineering methods. Our goal is to provide a methodology — or set of methodologies — to ensure usable and useful VE interfaces. Our work has produced a high-level methodology that is a logical progression of techniques, as detailed in Section 3.5.

Usability engineering, described rather simplistically, is the process by which usability is ensured for an interactive application, at all phases in the development process. These phases include user task analysis, user class analysis, design of the user interaction, rapid prototyping, user-centered evaluation, and iterative re-design based on evaluation results. Usability engineering includes both design and evaluations with users; it is not just applicable at the evaluation phase. Usability engineering is not typically structured hypothesis-testing-based experimentation.

Until recently, the terms "usability engineering" and "virtual environments" were rarely, if ever, used in the same sentence or context. VE developers have focused largely on producing the next new "gee whiz" gadget and "way cool" interaction technique with too little attention given to how users will benefit (or not) from those gadgets and techniques. Admittedly, "gee whiz, way cool" exploration in a new realm such as VEs is necessary. However, VEs have been rather mature for nearly a decade now, with an ever-expanding variety of possible applications.

Most extant usability engineering methods widely in current use were spawned by the development of GUIs. So even when VE developers attempt to apply usability engineering methods, most VE user interfaces are so radically different that well-proven techniques that produce usable GUIs may be neither particularly appropriate nor effective for VEs. Few principles for design of VE user interfaces exist, and almost none are empirically derived or validated. Use of usability engineering methods often results in VE designs that produce very unexpected reactions and performance of users, reaffirming the need for exactly such methods! Ultimately researchers and developers of VEs should seek to improve VE applications, *from a user's perspective* — ensuring their usability — by following a systematic approach to VE development such as offered from usability engineering methods.

During this first year of work, we have explored several usability engineering methods, including some adapted from GUI development, to see how successfully they can be applied to VE development. These methods include user task analysis, expert guidelines-based evaluation (also sometimes called heuristic evaluation or usability inspection), and formative usability evaluation. These methods, and their logical progression developing a VE, are

described in later sections. We conclude this report by presenting how this logical progression was applied to development of the Dragon VE.

2. SETTING THE CONTEXT FOR VE USABILITY ENGINEERING

Before we present specific usability engineering methods and give examples of their application, it is important to set the context for usability engineering. Developers of interactive systems — systems with user interfaces — often confuse the boundaries between the software engineering process and the usability engineering process. This is due at least in part to a lack of understanding of techniques for usability engineering, as well as which of these techniques is appropriate for use at which stages in the development process.

Software engineering has as a goal to improve software quality, but this goal, in and of itself, has little impact on *usability* of the resulting interactive system — in this case, a VE. For example, well-established "v&v" (validation and verification) techniques focus on software correctness, robustness, and so on, from a software developer's view, with little or no consideration of whether that software serves its users' needs. Thus, quality of the *user interface* — the usability — of an interactive system is largely independent of quality of the *software* for that system. Usability of the user interface is ensured by a user-centered focus on developing the *user interaction component*— the look and feel and behavior as a user interacts with an application. The user interaction component includes all icons, text, graphics, audio, video, and devices through which a user communicates with an interactive system. The user interaction component is developed by interaction designers and evaluators. Usability is unaffected by the *software component*, including that for both the user interface and the rest of the application (i.e., the non-user-interface software. Software engineers and systems engineers develop the software component of an interactive system.

Cooperation between usability engineers and software engineers is essential if VEs are to mature towards a truly user-centric work and entertainment experience. Thus, both the interaction component and the software component are necessary for producing any interactive system, including a VE, but the *component that ensures usability is the user interaction component.*

3. CURRENT USABILITY ENGINEERING METHODS

As mentioned, many of the techniques we are studying come from usability engineering methods for GUIs. But from our own studies, as well as from collaboration with and experiences of other VE researchers and developers, we have adapted GUI methods and produced some new methods for usability engineering of VE user interaction design. We have made adaptations and enhancements to existing methods at two levels to evolve a usability engineering methodology applicable to VEs: specific methods themselves had to be altered and extended to account for the complex interactions inherent in multimodal VEs, and various methods had to be applied in a meaningful sequence to both streamline the usability engineering process as well as provide sufficient coverage of the usability space.

To better understand the strengths and applicability of each individual GUI usability engineering method, we present a basic discussion of each method, to provide a brief overview of each. Following discussion of individual methods, we present benefits and insights gained from use of these adapted methods for VE user interaction development.

3.1. User Task Analysis

A *user task analysis* is the process of identifying a complete description of tasks, subtasks, and actions required to use a system as well as other resources necessary for user(s) and the system to cooperatively perform tasks (Hix & Hartson, 1993). User task

analyses follow a formal methodology, describing and assessing performance demands of user interaction and application objects. These demands are, in turn, compared with known human cognitive and physical capabilities and limitations, resulting in an understanding of the performance requirements of end users. User task analysis (Hackos & Redish, 1998) may be derived from several components of early systems analysis, and at the highest level, rely on an understanding of several physical and cognitive components. User task analyses are the culmination of insights gained through an understanding of user, organizational, and social workflow; needs analysis; and user modeling.

There are four generally accepted techniques for performing user task analysis: documentation review, questionnaire survey, interviewing, and observation (Eberts, 1999). Documentation review seeks to identify task characteristics as derived from technical specifications, existing components, or previous legacy systems. Questionnaire surveys are generally used to help evaluate interfaces that are already in use or have some operational component. In these cases, task-related information can be obtained by having domain experts such as existing users, trainers, or designers complete carefully designed surveys. Interviewing an existing or identified user base, along with domain experts and application "visionaries", provides very useful insight into what users need and expect from an application. Observation-based analysis, on the other hand, requires a user interaction prototype, resembling more the formative evaluation process than development of user task analysis, and as such, is used as a last resort. A combination of early analysis of application documentation and domain-expert- and user-interviewing typically provides the most useful and plentiful task analysis.

While user task analyses are typically performed early in the development process, it should be noted that — like all aspects of user interaction development — task analyses also need to be flexible and potentially iterative, allowing for modifications to performance and user interaction requirements during any stage of development. However, major changes to user task analysis during late stages of development can derail an otherwise effective development effort, and as such, should only be considered under dire circumstances.

User task analysis generates critical information used throughout all stages of the application development lifecycle. One such result is a top-down decomposition of detailed task descriptions. These descriptions serve, among other things, as an enumeration of desired functionality for designers and evaluators. Equally revealing results of task analysis include an understanding of required task sequences as well as sequence semantics. Thus, results of task analysis include not only identification and description of tasks, but ordering, relationships, and interdependencies among user tasks. These structured analytical results set the stage for other products of task analysis, including an understanding of information flow as users work through various task structures.

Another useful result of task analysis is indications of where and how users contribute information to, and are required to make decisions that influence, user task sequencing. This information, in turn, can help designers identify what part(s) of the tasking process can be automated by computer (one of the original and still popular services a computer may provide) affording a more productive and useful work environment.

Without a clear understanding of user task requirements, both evaluators and developers are forced to "best guess" or interpret desired functionality which inevitably leads to poor interaction design. Indeed, both user interaction and user interface software developers

claim that poor, incomplete, or missing task analysis is one of the most common causes of both poor software and product design.

3.2. Expert Guidelines-based Evaluation

Expert guidelines-based evaluation or *heuristic evaluation* or *usability inspection* aims to identify potential usability problems by comparing a user interaction design — either existing or evolving — to established usability design guidelines. The identified problems are then used to derive recommendations for improving interaction design. The method is used by usability experts to identify critical usability problems early in the development cycle so that design issues can be addressed as part of the iterative design process (Nielsen, 1994).

Expert guidelines-based evaluations rely on established usability guidelines to establish whether a user interaction design supports intuitive user-task performance (i.e., usability). (Nielsen, 1994) recommends three to five evaluators for a heuristic evaluation since fewer evaluators generally cannot identify enough problems to warrant the expense, and more evaluators produce diminishing results at higher costs. It is not clear whether this recommendation is cost-effective for VEs, since more complex VE interaction designs may require more evaluators than GUIs. Each evaluator first inspects the design alone, independently of other evaluators' findings. Results are then combined, documented, and assessed as evaluators communicate and analyze both common and conflicting usability findings.

A heuristic evaluation session may last one to two hours, or even more depending upon the complexity of the design. Again, VE interaction designs may require more time to fully explore. Further, heuristic evaluation can be done using a simple pencil and paper design (assuming that the design is mature enough to represent a reasonable amount of interaction components and interactions). This allows assessment to begin very early in the application development lifecycle.

The output from an expert guidelines-based evaluation should not only identify problematic interaction components and interaction techniques but should also indicate *why* a particular component or technique is problematic. Results from heuristic evaluation are subsequently used to remedy obvious and critical usability problems as well as to shape the design of subsequent formative evaluations (see Section 3.3). Evaluation results further serve as both a working instructional document for user-interface software developers, and more importantly, as fundamentally sound, research-backed, design rationale.

Given that expert guidelines-based evaluations are based largely on a set of usability heuristics, it can be argued that the evaluations are only as effective and reliable as the guidelines themselves. Nielsen (1994) presents usability heuristics for traditional GUIs. While these heuristics are considered to be the de facto standard for GUIs, we have found that they are too general, ambiguous, and high-level for effective and practical heuristic evaluation of VEs. Effectiveness is questioned on the simple fact that 3D, immersive, VE interfaces are much more complex than traditional GUIs. The original heuristics (implicitly) assume traditional input/output devices such as keyboard, mouse, and monitor, and do not address the appropriateness of VE devices such as CAVEs, HMDs, haptic and tactile gloves, various force feedback devices, etc. Determining appropriate VE devices for a specific application and its user tasks is critical to designing usable VEs. Practicality is questioned due to the abstractness of the heuristics; VE evaluators need specific, concrete guidelines to apply in rapid and reliable evaluations.

It is well-recognized that VE interfaces and VE user interaction are immature and currently emerging technologies for which standard sets of design, much less usability guidelines, do not yet exist. However, our recent research at Virginia Tech has produced a set of VE usability design guidelines, contained within a framework of usability characteristics (Gabbard & Hix, 1999). This framework document (see http://www.vpst.org/jgabbard/ve/framework) is available and provides a reasonable starting point for heuristic evaluation of VEs. The complete document contains several associated usability resources including specific usability guidelines, detailed context-driven discussion of the numerous guidelines, and citations of additional references.

The framework organizes VE user interaction design guidelines and the related context-driven discussion into four major areas: users and user tasks, input mechanisms, virtual model, and presentation mechanisms. The framework categorizes 195 guidelines covering many aspects of VEs that affect usability including navigation, object selection and manipulation, user goals, fidelity of imagery, input device modes and usage, interaction metaphors, and much more. The guidelines presented within the framework document are well-suited for performing heuristic evaluation of VE user interaction, since they provide both broad coverage of VE interaction/interfaces and are specific enough for practical application. For example, with respect to navigation within VEs, one guideline reads "provide information so that users can always answer the questions: Where am I now? What is my current attitude and orientation? Where do I want to go? How do I travel there?" Another guideline addresses methods to aid in usable object selection techniques stating "use transparency to avoid occlusion during selection."

We have successfully used these guidelines within context as a basis for heuristic evaluation of several VEs ranging from medical visualization to command and control situational awareness applications (Swartz et al., 1998; Hix et al., 1999b).

3.3. Formative Usability Evaluation

The term *formative evaluation* was coined by Scriven (1967) to define a type of evaluation that is applied during evolving or formative stages of design. Scriven used this in the educational domain for instructional design). Williges (1984) and Hix and Hartson (1993) extended and refined the concept of formative evaluation for the human-computer interaction domain.

The goal of formative evaluation is to assess, refine, and improve user interaction by iteratively placing representative users in task-based scenarios in order to identify usability problems, as well as to assess the design's ability to support user exploration, learning, and task performance (Hix & Hartson, 1993). Formative usability evaluation is an observational evaluation method which ensures usability of interactive systems by including users early and continually throughout user interface development. The method relies heavily on usage context (e.g., user task, user motivation, etc.) as well as a solid understanding of human-computer interaction (and in the case of VEs, human-VE interaction) and, as such, requires the use of usability experts (Hix & Hartson, 1993).

While the formative evaluation process was initially intended to support iterative development of instructional materials, it has proven itself to be a useful tool for evaluation of traditional GUI interfaces. Moreover, in the past few years, we have seen first hand evidence indicating that the formative evaluation process is also an efficient and effective method of improving the usability of VE interfaces (Hix et al., 1999a).

The steps of a typical formative evaluation cycle begin with development of user task scenarios, and are specifically designed to exploit and explore all identified task,

information, and work flows. Representative users perform these tasks as evaluators collect both qualitative and quantitative data. These data are then analyzed to identify user interaction components or features that both support and detract from user task performance. These observations are in turn used to suggest user interaction design changes as well as formative evaluation scenario and observation (re)design.

The formative evaluation process itself is iterative, allowing evaluators to continually refine user task scenarios in order to fine tune both the user interaction *and* the evaluation process. Contrary to popular belief, the formative evolution process produces both qualitative and quantitative results collected from representative users during their performance of task scenarios (del Galdo et al., 1986). One type of qualitative data collected is termed critical incidents (del Galdo et al., 1986; Hix & Hartson, 1993). A critical incident is a user event that has a significant impact, either positive or negative, on users' task performance and/or satisfaction (e.g., a system crash or error, being unable to complete a task scenario, user confusion, etc.). Critical incidents which have a negative effect on users' work flow can drastically impede usability and may even have a dramatic effect on users' perceptions of application quality, usefulness, and reputation. As such, any obvious critical incidents are best discovered during formative evaluation phases as opposed to consumers' desktops.

Equally important are the quantitative data collected during formative evaluation. These data include measures such as how long it takes a user to perform a given task, the number of errors encountered during task performance, etc. Collected quantitative data are then compared to appropriate baseline metrics, sometimes initially redefining or altering evaluators' perceptions of what should be considered baseline. Both qualitative and quantitative data are equally important since they each provide unique insight into an interaction design's strengths and weaknesses.

3.4. Summative Evaluation

Summative evaluation, in contrast to formative evaluation, is typically performed after a product or design is more or less complete; its purpose is to statistically compare several different systems, for example, to determine which one is "better" — where better is defined in advance. Another goal of summative evaluation is to measure and subsequently compare the productivity and cost benefits associated with various designs. In this fashion, evaluators are simply comparing the best of a few refined designs to determine which of the "finalists" is best suited for delivery.

The term summative evaluation was also coined by Scriven (1967), again for use in the instructional design field. As with the formative evaluation process, human-computer interaction researchers (e.g., (Williges, 1984)) have applied the theory and practice of summative evaluation to interaction design with surpassingly successful results.In practice, summative evaluation can take on many forms. The most common are the comparative, field trial, and more recently, the expert review (Stevens et al., 1997). While both the field trial and expert review methods are well-suited for instructional content and design assessment, they typically involve assessment of single prototypes or field-delivered designs. In the context of VE design, we are mostly interested in assessing the quality of two or more user interaction designs, and as such, have focused on the comparative approach. Our experiences have found that this approach is very effective for analyzing the strengths and weaknesses of various well-formed, completed designs using representative user scenarios.

(Stevens et al., 1997) presents a short list of questions that summative evaluation should address. We have modified these questions to address summative, comparative evaluation of VE user interfaces. The questions include:

- What are the strengths and weaknesses associated with each user interaction design?
- To what extent does each user interaction design support overall user and system goal(s)?
- Did users perceive increased utility and benefit from each design? In what ways?
- What components of each design were most effective?
- Was the user interaction evaluation effort successful? That is, did it provide a cost-effective means of improving design and usability?
- Were the results worth the program's cost?

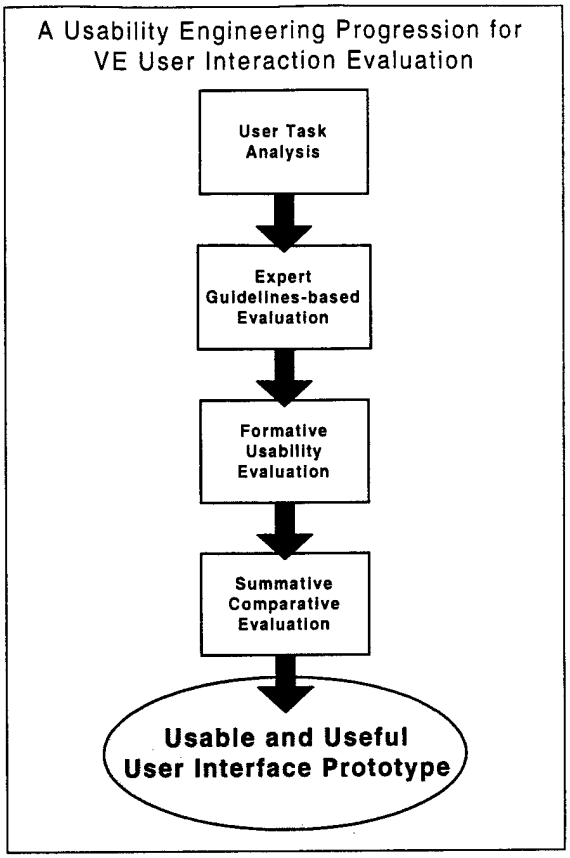


Figure 1. Successful progression of usability engineering methods.

3.5. A Successful Progression

As previously mentioned, one of our long-term research goals is to produce methodologies to improve the usability of VE user interaction designs. More specifically, the goal is to develop, modify, and fine-tune usability engineering techniques specifically for VEs. Techniques to aid in usability engineering of VEs will mature, and as such, the potential for conducting effective evaluations and delivering subsequent usable and useful VEs will increase.

Our current efforts are focusing on the combination of usability engineering techniques described in Sections 3.1 to 3.4. As depicted in Figure 1, our applied research over the past several years has shown that, at a high level, progressing from user task analysis to expert guidelines-based evaluation to formative evaluation to summative evaluations is an efficient and cost-effective strategy for designing, assessing, and improving the user interaction — usability engineering — of a VE.

One of the strengths of this progression is the fact that it exploits a natural ordering or evolution of interaction design and prototyping, with each method generating a streamlined set of information the next method utilizes. In this sense, each method is able to generate a much better starting point for subsequent methods, than when applied in a standalone fashion. Moreover, simply applying more than one usability engineering method ensures more complete coverage of an interaction design's "usability space", each revealing its niche of particular usability problems, collectively shaping a more

usable VE. And finally, the progression of methods also produces a "paper trail" of persistent documentation that may serve as documented design rationale.

This progression is very cost-effective for assessing and improving VEs. For example, summative studies are often performed on VE interaction designs that have had little or not task analysis or expert guidelines-based or formative evaluation. This may result in a situation where the expensive summative evaluation is essentially comparing "good apples" to "bad oranges" (Hix et al., 1999). Specifically, a summative study of two different VEs may be comparing one design that is inherently better, in terms of usability, than the other one. When all designs in a summative study have been developed following our suggested progression of usability engineering, then the comparison is more valid. Experimenters will then know that the interaction designs are basically equivalent in terms of their usability, and any differences found among compared designs are, in fact, due to variations in the fundamental nature of the designs, and not their usability.

4. APPLICATION OF USABILITY ENGINEERING METHODS TO DRAGON

Research and development performed both in our own labs and in other VE labs has shown the usability engineering techniques just described to be effective for ensuring usability. The Dragon case study presents the application of these methods to a military command and control application developed for the Responsive Workbench at the Naval Research Laboratory's Virtual Reality Laborator in Washington DC. For development of the Dragon VE application, we are following the usability engineering methods described above with great success in evolving the interaction design.

4.1. Dragon

Personnel at NRL, in collaboration with Virginia Tech researchers, have developed Dragon, a VE for battlefield visualization (Hix et al., 1999a). Implemented on a Responsive Workbench, Dragon's metaphor for visualizing and interacting with 3D computer-generated scenery uses a familiar tabletop environment. Applications in which several users collaborate around a work area, such as a table, are excellent candidates for the Workbench. This metaphor is especially familiar to Naval personnel and Marines, who have, for decades, accomplished traditional battlefield visualization on a tabletop. Paper maps of a battlespace are places under sheets of acetate. As intelligence reports arrive from the field, technicians use grease pencils to mark new information on the acetate. Commanders then draw on the acetate to plan and direct various battlefield situations. Historically, before high-resolution paper maps, these operations were performed on a sandtable, literally a box filled with sand shaped to replicate battlespace terrain. Commanders moved around small physical replicas of battlefield objects to direct battlefield maneuvers. The fast-changing modern battlefield produces so much time-critical information that these cumbersome, time-consuming methods are inadequate for effectively visualizing and commanding a modern-day battlespace.

Dragon, shown in Figure 2, was developed on the Responsive Workbench to give a 3D display for observing and managing battlefield information shared among technicians and commanders. Visualized information includes a high-resolution terrain map; entities representing friendly, enemy, unknown, and neutral units; and symbology representing other features such as obstructions or key map points. Users can navigate to observe the map and entities from any angle and orientation around the Workbench.

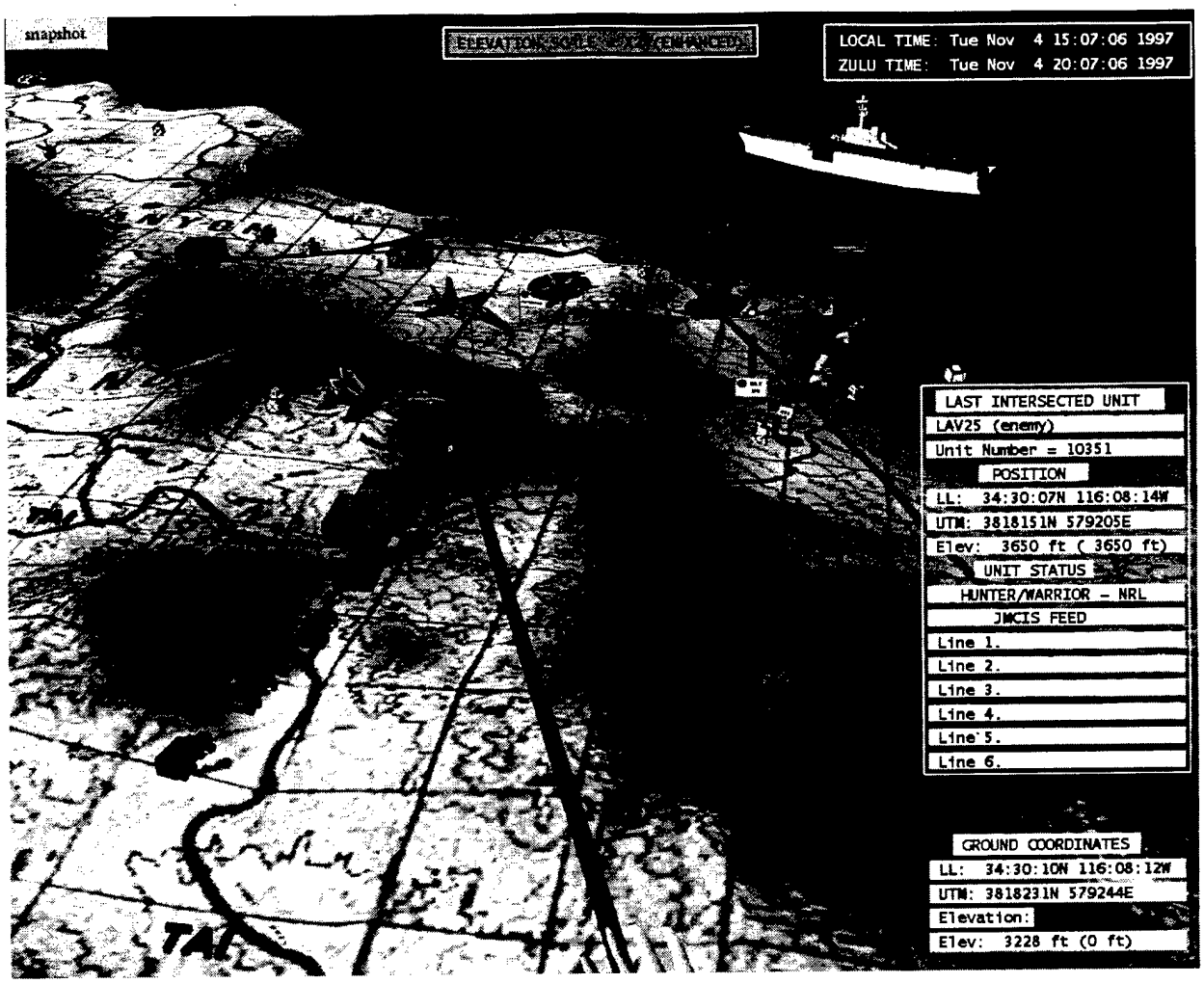


Figure 2: Screen shot from the Dragon battlefield visualization virtual environment.

Dragon's early development was based on an admittedly cursory user task analysis, which drove early design. This was followed, however, by numerous cycles of expert guidelines-based evaluation as well as formative evaluations. Early in Dragon design, we produced and assessed three general interaction methods for the Workbench, any of which could have been used to interact with Dragon: hand gestures using a pinchglove (Obeysekare et al., 1996), speech recognition, and a hand-held flightstick. Although it was an interesting possibility for VE interaction, our formative evaluations found that speech recognition is still too immature for battlefield visualization. We further found the pinchglove to be too fragile and time-consuming to pass from user to user around the Workbench. It also worked best for right-handed users whose hands were approximately the same size. In contrast, our formative evaluations revealed that the flightstick was robust, easily handed from user to user, and worked for both right- and left-handed users.

Based on these formative evaluations, we modified a three-button game flightstick by removing its base and placing a six degree-of-freedom position sensor inside. Our initial designs used a laser pointer metaphor in which a laser beam appeared to come out the "head" of the flightstick as a user pointed it toward the VE map. When a beam intersected terrain or an object, a highlight marker appeared.

In early demonstrations of initial versions of Dragon in real military exercises for battlefield planning, users indicated they found Dragon's accurate and current visualization of the battlespace to be more effective and efficient than the traditional method of maps, acetate, and grease pencils. Following these successful demonstrations and positive feedback, we began intensive usability engineering of Dragon's user interaction design.

Also during our early demonstrations and evaluations, we observed that navigation – how users manipulate their viewpoint to move from place to place in a virtual world (in this case, the battlefield map) – profoundly affects all other user tasks. If a user cannot successfully navigate to move about in a virtual world, then other user tasks such as selecting an object or grouping objects cannot be performed. A user cannot query an object if the user cannot navigate through the virtual world to get to that object. Although we performed a user task analysis before our guidelines-based and formative evaluations, these evaluations supported our expectations of the importance of navigation.

Expert guidelines-based evaluation was done extensively for Dragon, prior to much formative evaluation. However, we were developing the framework of usability characteristics of VEs at the same time we were performing the guidelines-based evaluation, and so the guidelines we used were much more ad hoc and less structured than those that eventually became the framework. Nonetheless, even our informal inspection of the evolving Dragon interaction design provided tremendous feedback in what worked and what didn't, especially for use of the wand and other aspects of navigation in Workbench VE applications in general and Dragon in particular. During these evaluations, VE user interaction design experts worked alone or collectively to assess the Dragon interaction design.

In our earliest evaluations, the experts did not follow specific user task scenarios per se, but engaged simply in "free play" with Dragon using the wand. All experts knew enough about the purpose of Dragon as a battlefield visualization VE to explore the kinds of tasks that would be more important for Dragon users. During each session, one person was typically "the driver", holding the flightstick and generally deciding what and how to explore in the application. One and sometimes two other experts were observing and commenting as the "driver" worked, with much discussion during each session. Major design problems that were uncovered in our expert guidelines-based evaluation of Dragon included poor mapping of navigation tasks (i.e., pan, zoom, pitch, heading) to flightstick buttons, missing functionality (e.g., exocentric rotate, terrain following), problems with damping of map movement in response to flightstick movement, and graphical and textual feedback to the user about the current navigation task. After these evaluations had revealed and remedied as many design flaws as possible, we moved on to formative evaluation.

We used the basic Dragon application to perform extensive evaluations, using anywhere from one to three users for each cycle of evaluation. From a single evaluation session we often uncovered design problems so serious that it was pointless to have a different user attempt to perform scenarios with the same design. So we would iterate the design, based on our observations, and begin a new cycle of evaluation. We went through four major cycles of iteration in all.

In designing our scenarios for formative evaluation, we carefully considered coverage of specific usability issues related to navigation. For example, some of the tasks exploited an egocentric (user moves through the virtual world) navigation metaphor, while others exploited an exocentric (user moved the world) navigation metaphor. Some scenarios

exercised various navigation tasks (e.g., degrees of freedom: pan, zoom, rotate, heading, pitch, roll) in the virtual map world. Other scenarios served as primed exploration or non-targeted searches for specific features or objects in the virtual world. Still others were design to evaluation rate control versus position control.

During each of six formative evaluation sessions, we first asked the participant to play with the flightstick to figure out which button activated which navigation task. We timed each user as they attempted to determine this, and took notes on comments and any critical incidents that occurred. Once a user had successfully figured out how to use the flightstick, we began having them formally perform the task scenarios. Only one user was unable to figure out the flightstick in less than 15 minutes; we told this user details they had not yet discovered and proceeded with the scenarios.

Time to perform the set of scenarios ranged from about 20 minutes to more than one hour. We timed user performance of individual tasks and scenarios, and counted errors made during task performance. A typical error was moving the flightstick in the wrong direction for the particular navigation metaphor (exocentric or egocentric) that was currently in use. Other errors involved simply not being able to maneuver the map (e.g., to rotate it) and persistent problems with mapping navigation tasks (degrees of freedom) to flightstick buttons, despite our extensive prior evaluations to minimize this issue. During each formative evaluation session, we had at least two and often three evaluators present. One served as the facilitator to interact with the participant and keep the session moving; the other one or two evaluators recorded times, counted errors, and collected critical incidents and other qualitative data. While these sessions (like those of Crumbs in the CAVE) seem personnel-intensive, with two or three evaluators involved, we found that the quality and quantity of data collected by multiple evaluators greatly outweighed the cost of those evaluators. A surprising amount of our effort was spent on mapping flightstick buttons to navigation tasks (pan, zoom, rotate, heading, pitch, roll), but we found it paid off with more effective, intuitive mappings.

As mentioned earlier, we went through four major iterations of the Dragon interaction design, based on our evaluations. The first iteration, the "Virtual Sandtable," was an egocentric navigation metaphor based on the sandtable concept briefly described earlier. This was the version demonstrated in the military exercises also mentioned previously. A key finding of this iteration was that users wanted a terrain-following capability, allowing them to "fly" over the map – an egocentric design. Map-based navigation worked well when globally manipulating the environment and conducting operations on large-scale units. However, for small-scale operations, users wanted this "fly" capability to visually size up terrain features, entity placement, fields of fire, lines of sight, etc.

The second iteration, "Point and Go," used the framework of usability characteristics of VEs to suggest various possibilities for an egocentric navigation metaphor design. This metaphor attempted to avoid having different modes (and flightsick buttons) for different navigation tasks because of known usability problems with moded interaction. Further, we based this decision on how a person often navigates to an object or location in the real world; namely, they point (or look) and then go (move) there. Our reasoning was that adopting this same idea to egocentric navigation would simplify the design and at least loosely mimic the real world. So in this design, a user simply pointed the flightstick toward a location or object of interest, and pressed the trigger to fly there. We found that the single gesture to move about was not powerful enough to support the diverse, complicated variety of navigation tasks

inherent in Dragon. Furthermore, a single gesture meant that all degrees of freedom were controlled by that single gesture. This resulted in, for example, unintentional rolling when a user only wanted to pan or zoom. Essentially we observed a control versus convenience trade-off. Many navigation tasks (modes) were active simultaneously, which was convenient but difficult to physically control for a user. With separate tasks (modes) there was less convenience but physical control was easier because degrees of freedom were more limited in each mode. In addition to these serious problems, we found that users wanted to rotate around an object, such as to move completely around a tank. This indicated that Dragon needed an exocentric rotate ability, which we added. This interesting finding showed that neither a pure egocentric nor a pure exocentric metaphor was desirable; each metaphor has aspects that are more or less useful depending on user goals. While this may seem obvious, it was confirmed by users through our formative evaluation approach. Further, the somewhat poor performance of what we thought was the natural "point and go" metaphor was rather counterintuitive, and further demonstrates that mimicking VE design after the real world is not always successful from the VE user's perspective.

The third iteration, "Modal," went from the extreme of all navigation tasks coupled on a single button as in the previous iteration to a rather opposite design in which each navigation task was a separate mode. Specifically, as a user clicked the left or right flightstick button, Dragon cycled successively through the tasks of pan, zoom, pitch, heading, and exocentric rotate. A small textual indicator was displayed on the Workbench to show the current mode. Once a user had cycled to the desired task, the user simply moved the flightstick and that task was enabled with no need to push any flightstick button. We observed that, as we expected, it was very cumbersome for users to always have to cycle between modes, and it was obvious that we still had not achieved a compromise between convenience and control.

In our fourth iteration of the Dragon interaction design, "Integrated Navigation," based on the framework of usability characteristics of VEs and our own user observations of what degrees of freedom could be logically coupled in the Dragon application, we produced a hybrid design of the modeless/moded designs of prior iterations. Specifically, we coupled pan and zoom onto the flightstick trigger, pitch and heading onto the left flightstick button, and exocentric rotate and zoom onto the right flightstick button. This fourth generation interaction design for Dragon finally achieved the desired convenience versus control compromise. In our final formative evaluation studies, we found that at last we had a design for navigation that seemed to work well. The only usability problem we observed was minor: damping of map movement was too great and needed some adjustment, which we made.

4.2. Dragon Summative Evaluations

As the formative evaluation cycles neared completion for Dragon, we began planning summative studies for our navigation design. Design parameters that affect usability of the VE navigation metaphor to be studied in our summative evaluations are those that, in general, could not be decided from the formative evaluations. We identified 27 such design parameters, shown in Table 1, that potentially affect the usability of user navigation. The organization of these parameters is based on the framework of usability characteristics of VEs. The four main areas of this framework are the bold headings in Table 1, and our design parameters for navigation are grouped based on these areas.

Based on the framework of usability characteristics, observations during our evaluations, extensive literature review, and our expertise in VE interaction design, we have narrowed

the numerous variables to four that we feel are most critical for navigation, and therefore most important for the first cycle of summative evaluations. These four variables (included in Table 1) and their level of treatment in our planned summative study are:

- navigation metaphor (ego- versus exocentric)
- gesture control (rate versus position of hand movement)
- visual presentation device (Workbench, desktop, CAVE)
- stereopsis (present versus absent)

This summative study is currently designed, and will be performed as soon as the Dragon application runs on all three presentation devices. It is, we believe, one of the first formal studies to directly compare three different VE devices using the same application and the same user tasks.

Design parameters for navigation evaluation

User Tasks	Input Devices	Virtual Model	Presentation Devices
user scenarios	Navigation viewpoint	mode switching	visual presentation device
navigation presets	Navigation degrees-of-freedom	mode feedback	stereopsis
	gestures to trigger actions	number of modes	
	speech input	visual navigation aids	
	number of flightstick buttons	dataset characteristics	
	input device type	visual terrain representatio	
	Movement deadspace	visual (battlefield) object representation	
	Movement damping	visual input device representation	
	user gesture work volume	size of (battlefield) objects	
	gesture mapping	visual object relationship representation	
	button mapping	map constrained vs. floating	
	head tracking		

Table 1. Design parameters for navigation, organized by framework of usability characteristics for VEs (Gabbard and Hix, 1999)

5. CONCLUSION

Despite improvement in efforts to improve usability engineering of VEs, there still is not nearly enough usability engineering applied during development of VEs. And there is still a need more cost-effective, high-impact methods. As VEs become more mature, established interaction techniques and generic tasks for VEs will emerge, and have the potential — as they did with GUIs — to improve usability. But the design space and options for VEs is enormously greater than that for GUIs, and VE applications tend to be much more complex than many GUIS. So even "standard" interaction techniques, devices, and generic tasks for VEs will help improve usability only by a small fraction. Usability engineering will continue to be a necessary process if new and exciting VEs that are, in fact, usable and useful for their users are to be created.

7. REFERENCES

Eberts, R. E., Unpublished lecture on task analysis: IE486 Work Design and Analysis II, Purdue University (1999). Presentation is online at http://palette.ecn.purdue.edu/~ie486/Class/Lecture/lect14/sld001.htm

Gabbard, J. & Hix, Deborah. (1999). Usability Engineering for Virtual Environments through a Taxonomy of Usability Characteristics. Currently submitted to Presence: Teleoperators and Virtual Environments.

del Galdo, E.M., Williges, R.C., Williges, B.H., & Wixon, D.R. (1986). An Evaluation of Critical Incidents for Software Documentation Design, In Proceedings of Thirtieth Annual Human Factors Society Conference, Anaheim, CA.

Hackos, J. T. & Redish, J. C. (1998) User and Task Analysis for Interface Design. John Wiley & Sons, Inc. New York.

Hix, D. & Hartson, H. R. (1993). Developing User Interfaces: Ensuring Usability through Product & Process, John Wiley and Sons, Inc.

Hix, D., Swan, E. J., Gabbard, J. L., McGee, M., Durbin, J., & King, T. (1999a). User-centered design and evaluation of a real-time battlefield visualization virtual environment. In Proceedings of the IEEE VR'99 Conference. (NOTE: This paper was awarded Best Technical Paper at the VR'99 Conference.)

Hix, D., Amento, B., Templeman, J. N., Schmidt-Nielsen, A., & Sibert, L. (1999b) An Empirical Comparison of Interaction Techniques for Panning and Zooming in Desktop Virtual Environments. Sumbitted to Journal of Human Factors.

Nielsen, J. (1994). Heuristic evaluation. In Usability Inspection Methods, chapter 2, pp. 25-62. John Wiley & Sons.

Obeysekare, U., Williams, C., Durbin, J., Rosenblum, L., Rosenberg, R., Grinstein, F. , Ramamurthi, R., Landsberg, A. & Sandberg, W. (1996). Virtual Workbench: A Non-Immersive Virtual Environment for Visualizing and Interacting with 3D Objects for Scientific Visualization. In Proceedings of IEEE Visualization'96, IEEE Computer Society Press, pp. 345-349.

Scriven, M. (1967). The methodology of evaluation. In R. E. Stake (Ed.), Perspectives of curriculum evaluation, American Educational Research Association Monograph. Chicago: Rand McNally.

Stevens, F., L. Frances, and L. Sharp. (1997). User-friendly handbook for project evaluation: science, mathematics, engineering, and technology education. NSF 93-152.

Swartz, K., Thakkar, U., Hix, D., & Brady, R. (1999). Evaluating the Usability of Crumbs: A Case Study of VE Usability Engineering Methods. to appear in Proc. Third International Immersive Projection Technology Conference, Stuttgart, Germany.

Williges, R. C. (1984). Evaluating Human-Computer Software Interfaces. In Proceedings of International Conference on Occupational Ergonomics.

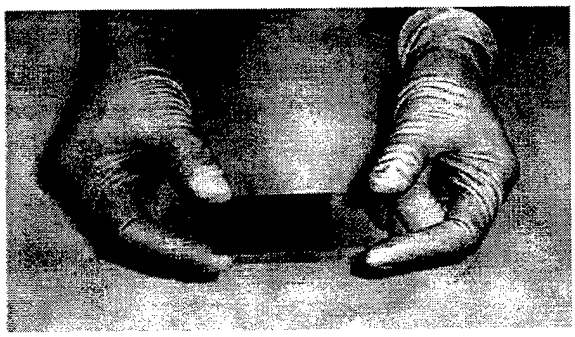
2.7 Degradation of fiber optic system under dynamic use and fabrication and test of large area array (R.O. Claus and Staff- Task 1.4b.3) CDRL Data Item 0026, 0027

1.0 Tests of Mechanically-Flexible Display Arrays Formed by ESA Processing

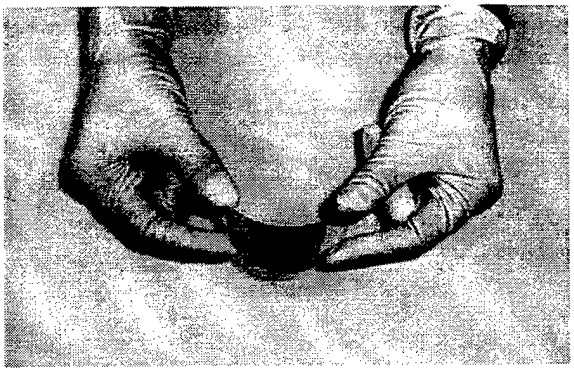
During the program months, we have mechanically tested the robustness of the mechanically flexible optoelectronic display demonstrator article that has been described in detail in prior monthly reports. The display element was formed through the self-assembly of PPV and other molecular precursors, using the process detailed in those reports.

The objective of our mechanical testing has been to investigate how flexing may degrade the performance of either the mechanical actuation behavior of the thin films, or the optical light emitting properties. First, as shown in Figure 1, ESA-formed active polymer thin films on mechanically flexible ITO-coated polyester substrates were repeatedly bent by hand to form U-shaped geometries, but without creasing.

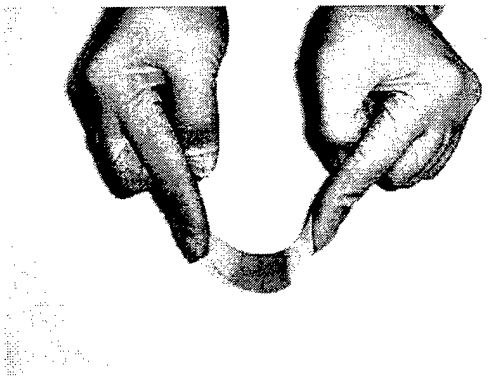
Measurements of several properties prior to and after flexing, including UV-vis absorption, film thickness as measured by multi-wavelength ellipsometry, and simple visual observation, were used to observe potential variations in material quality. No variations were observed for six samples flexed as shown for 50 cycles. Atomic force microscope images obtained prior to and after flexing did not display any differences.



(a)



(b)



(c)

Figure 1. Mechanical Testing of Active Device Polymer Formed by ESA Process on Flexible Polyester Substrate.

Additional testing of LED arrays formed from such deposited films is shown in Figure 2. Here, the qualitative "tape test" was used to verify good mechanical adhesion between the deposited thin film layers and the substrate, and good mechanical integrity of the film itself. Transparent adhesive tape was attached to LED array devices as shown in Figure 2 (a), and finger pressure was used to carefully eliminate the formation of air bubbles between the adhesive and the multilayered thin film on the substrate. Testing was performed by pulling the tape vertically with respect to the plane of the substrate, as shown in Figure 2 (b). Visual inspection and microscopy did not reveal that any areas were removed from the surface of the specimen, and characterization using UV-vis and ellipsometry did not indicate the existence any variations between measurements made prior to or after testing.

2.8 Specification of design elements for intranet (Ken Reifsnider and Rick Habayeb- Task 1.5)
CDRL Data Item 0028,

A ship information management system (SIMS) is needed to optimally tie ship systems together as an intranet. A three-tier architecture is planned to provide connectivity, scalability, interoperability, and flexibility of SIMS. The three tiers are the client or presentation tier, a middle tier or application logic tier and a databases tier. The middle tier is implemented by enterprise application servers. The Ship/enterprise Application Servers (SAS) are software platforms that allow a developer to launch intranet/internet applications. An application server provides a platform on which one can run middle-tier business/ship logic. These servers are designed to support applications with transaction

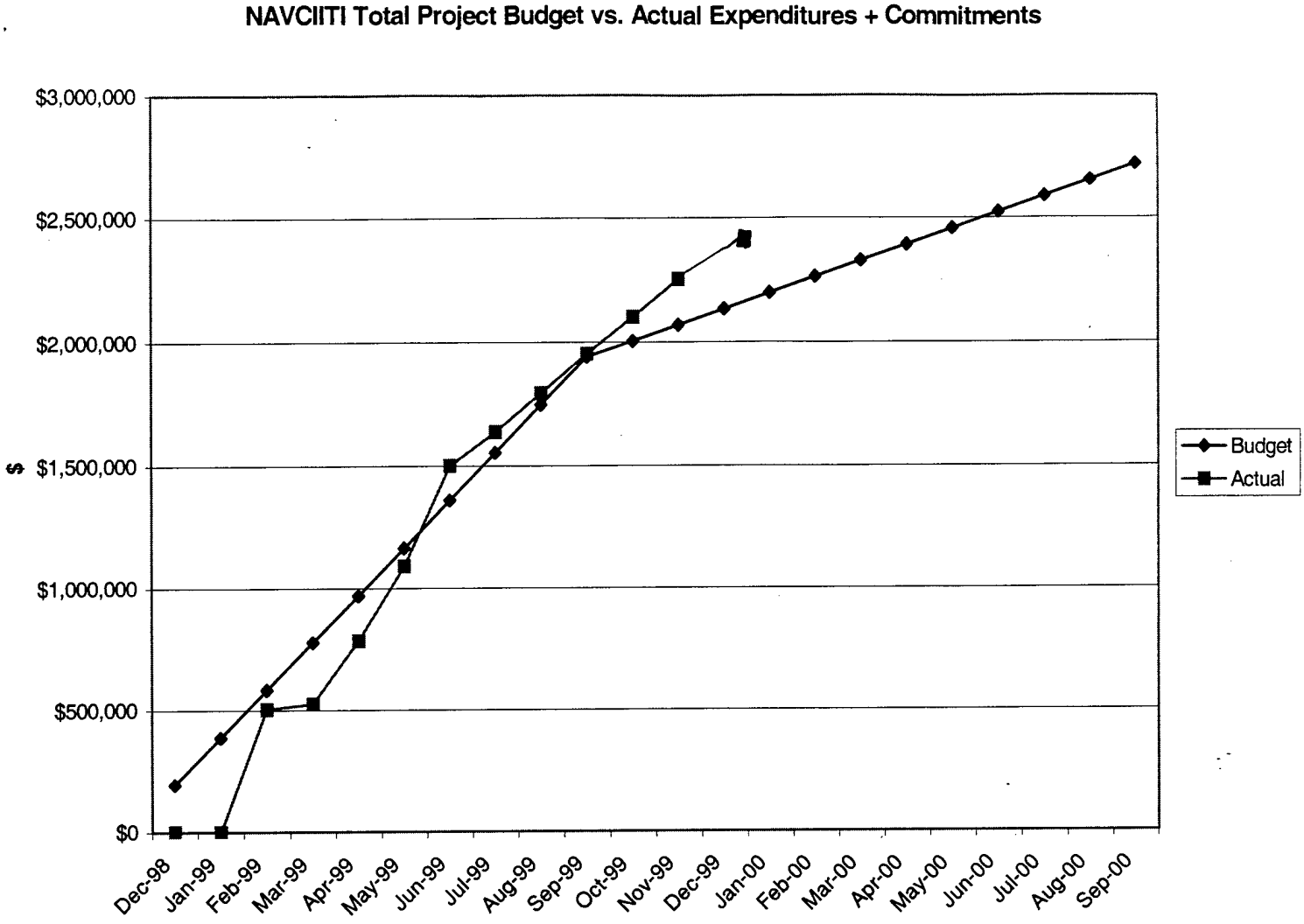
processing, or with complex decision processing requirements such as passing targeting/tracking information and mission planning data. SIMS provides the intrinsic capability of generating and distributing information in the network, e.g., targeting, mission planning, and command and control information. Networking SIMS with the VON will provide the desired interoperability realism in a network centric environment. Computer network interoperability hardware, and network system interoperability are projects that address the linkages of the network and its IPs. Through SIMS, network interoperability, wireless and mobile communication, basic and realistic modeling of the network traffic and security can be optimized and evaluated. Profiling and Quality of Service management metrics will be used to evaluate some of the effectiveness parameters of the network. Visualization, human interaction, and collaboration tools will be used to optimize and evaluate the NAVCIITI architecture and configuration. Ultimately, through the CAVE, the NAVCIITI test bed can simulate "ship under attack" or a ship in "self-defense" mode. These simulations will build on the ship performance models that reside in the databases. Displays are very important in the sensors and communication hardware. Fiber optics LAN are very desirable to cope with the harsh EMI / RFI environment aboard ships. Fiber optics LAN provides a huge bandwidth for optimal distribution of data and imaging information.

2.9 Description of features of scalable VE Interfaces (Ron Kriz – Task 1.6.2). CDRL Data Item 0032 A prototype system that would allow participants at remote sites to collaborate in immersive environments was created and tested. The prototype system is called the "Collaborative CAVE Console" (CCC). The CCC was based on CAVERN soft and LIMBO. System was demonstrated at the NCSA Access Center on November 16, 1999 as part of Supercomputing 99 demonstration of collaboration over high speed networks. Four workstations, each running CCC independently were linked with "Network Virginia" to remote sites at Lynchburg, Shenandoah, and Arlington, Virginia. At the NCSA Access Center at Arlington, Virginia, the CCC was running on a SGI Octane and desk side Power Onyx that projected images onto an I-Desk ("one-walled CAVE"). Although voices could be linked via the internet, in this demonstration voices were linked using a standard telephone communication conference call. This same system was also demonstrated on September 22, 1999 at the Hart Congressional Office Building in Washington D.C. where two SGI Octanes, located at the Hart Office Bldg., were linked to the CAVE at Virginia Tech. We also plan on demonstrating the CCC at the Inernet2 Studio in Richmond Virginia in the near future.

Scaling the CCC to the large scale CAVE was not possible because the large scaled CAVE structure was not built. The extension of CCC to the General Dynamics "Command Post "of the Future was also not realized because this project was not funded.

1)Design requirements for high speed distributed collaborative immersive VEs, and 2) downloadable prototype of collaborative immersive tools used between remote sites are posted on the web page: http://www.sv.vt.edu/future/cave/software/ccc/

Test and evaluation of prototype usefulness with recommendations for full scale system is work-in-progress.



3.0 Financial Status